A. Yogurt Sale

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

The price of one yogurt at the "Vosmiorochka" store is a burles, but there is a promotion where you can buy two yogurts for b burles.

Maxim needs to buy **exactly** n yogurts. When buying two yogurts, he can choose to buy them at the regular price or at the promotion price.

What is the minimum amount of burles Maxim should spend to buy n yogurts?

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first and only line of each test case contains three integers n, a, and b ($1 \le n \le 100$, $1 \le a, b \le 30$) — the number of yogurts Maxim wants to buy, the price for one yogurt, and the price for two yogurts on promotion.

Output

For each test case, print in a separate line the minimum cost of buying n yogurts at "Vosmiorochka".

Standard Input	Standard Output
4 2 5 9 3 5 9 3 5 11 4 5 11	9 14 15 20

Note

In the third test case of the example, it is more advantageous to buy three yogurts for 15 burles than two for 11 and one for 5.

In the fourth test case of the example, you need to buy four yogurts, each for 5 burles.

,

B. Progressive Square

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

A *progressive square* of size n is an $n \times n$ matrix. Maxim chooses three integers $a_{1,1}$, c, and d and constructs a *progressive square* according to the following rules:

$$a_{i+1,j} = a_{i,j} + c$$

$$a_{i,j+1} = a_{i,j} + d$$

For example, if $n = 3$, $a_{1,1} = 1$, $c = 2$, and $d = 3$, then the *progressive square* looks as follows:

$$\begin{pmatrix} 1 & 4 & 7 \\ 3 & 6 & 9 \\ 5 & 8 & 11 \end{pmatrix}$$

Last month Maxim constructed a *progressive square* and remembered the values of n, c, and d. Recently, he found an array b of n^2 integers in random order and wants to make sure that these elements are the elements of **that specific** square.

It can be shown that for any values of n, $a_{1,1}$, c, and d, there exists exactly one *progressive square* that satisfies all the rules.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains three integers n, c, and d ($2 \le n \le 500$, $1 \le c, d \le 10^6$) — the size of the square and the values of c and d as described in the statement.

The second line of each test case contains $n \cdot n$ integers $b_1, b_2, \ldots, b_{n \cdot n}$ ($1 \le b_i \le 10^9$) — the elements found by Maxim.

It is guaranteed that the sum of n^2 over all test cases does not exceed $25 \cdot 10^4$.

Output

For each test case, output "YES" in a separate line if a *progressive square* for the given n, c, and d can be constructed from the array elements a, otherwise output "NO".

You can output each letter in any case (lowercase or uppercase). For example, the strings "yEs", "yes", "Yes", and "YES" will be accepted as a positive answer.

Standard Input	Standard Output
5 3 2 3 3 9 6 5 7 1 10 4 8 3 2 3 3 9 6 5 7 1 11 4 8 2 100 100	NO YES YES NO NO

400 300 400 500 3 2 3 3 9 6 6 5 1 11 4 8 4 4 4 15 27 7 19 23 23 11 15 7 3 19 23 11 15 11 15	

,

C. Inhabitant of the Deep Sea

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

n ships set out to explore the depths of the ocean. The ships are numbered from 1 to n and follow each other in ascending order; the i-th ship has a durability of a_i.

The Kraken attacked the ships k times in a specific order. First, it attacks the first of the ships, then the last, then the first again, and so on.

Each attack by the Kraken reduces the durability of the ship by 1. When the durability of the ship drops to 0, it sinks and is no longer subjected to attacks (thus the ship ceases to be the first or last, and the Kraken only attacks the ships that have not yet sunk). If all the ships have sunk, the Kraken has nothing to attack and it swims away.

For example, if $n = 4$, $k = 5$, and $a = [1, 2, 4, 3]$, the following will happen:

1. The Kraken attacks the first ship, its durability becomes zero and now $a = [2, 4, 3]$;
2. The Kraken attacks the last ship, now $a = [2, 4, 2]$;
3. The Kraken attacks the first ship, now $a = [1, 4, 2]$;
4. The Kraken attacks the last ship, now $a = [1, 4, 1]$;
5. The Kraken attacks the first ship, its durability becomes zero and now $a = [4, 1]$.

How many ships were sunk after the Kraken's attack?

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and k ($1 \le n \le 2 \cdot 10^5$, $1 \le k \le 10^{15}$) — the number of ships and how many times the Kraken will attack the ships.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the durability of the ships.

It is guaranteed that the sum of n for all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output the number of ships sunk by the Kraken on a separate line.

Standard Input	Standard Output
6 4 5 1 2 4 3 4 6 1 2 4 3 5 20 2 7 1 8 2 2 2 3 2	2 3 5 0 2 2

2 15 1 5 2 7 5 2	

,

D. Inaccurate Subsequence Search

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

Maxim has an array a of n integers and an array b of m integers ($m \le n$).

Maxim considers an array c of length m to be good if the elements of array c can be rearranged in such a way that at least k of them match the elements of array b.

For example, if $b = [1, 2, 3, 4]$ and $k = 3$, then the arrays $[4, 1, 2, 3]$ and $[2, 3, 4, 5]$ are good (they can be reordered as follows: $[1, 2, 3, 4]$ and $[5, 2, 3, 4]$), while the arrays $[3, 4, 5, 6]$ and $[3, 4, 3, 4]$ are not good.

Maxim wants to choose every subsegment of array a of length m as the elements of array c. Help Maxim count how many selected arrays will be good.

In other words, find the number of positions $1 \le l \le n - m + 1$ such that the elements $a_l, a_{l+1}, \ldots, a_{l+m-1}$ form a good array.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains three integers n, m, and k ($1 \le k \le m \le n \le 2 \cdot 10^5$) — the number of elements in arrays a and b, the required number of matching elements.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$) — the elements of array a. Elements of the array a are not necessarily unique.

The third line of each test case contains m integers $b_1, b_2, \ldots, b_m$ ($1 \le b_i \le 10^6$) — the elements of array b. Elements of the array b are not necessarily unique.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$. Similarly, it is guaranteed that the sum of m over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output the number of good subsegments of array a on a separate line.

Standard Input	Standard Output
5 7 4 2 4 1 2 3 4 5 6 1 2 3 4 7 4 3 4 1 2 3 4 5 6 1 2 3 4 7 4 4 4 1 2 3 4 5 6 1 2 3 4 11 5 3 9 9 2 2 10 9 7 6 3 6 3	4 3 2 4 1

6 9 7 8 10 4 1 1 4 1 5 6 6	

Note

In the first example, all subsegments are good.

In the second example, good subsegments start at positions 1, 2, and 3.

In the third example, good subsegments start at positions 1 and 2.

,

E. Long Inversions

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   256 megabytes
```

A binary string s of length n is given. A binary string is a string consisting only of the characters '1' and '0'.

You can choose an integer k ($1 \le k \le n$) and then apply the following operation any number of times: choose k consecutive characters of the string and invert them, i.e., replace all '0' with '1' and vice versa.

Using these operations, you need to make all the characters in the string equal to '1'.

For example, if $n = 5$, $s = 00100$, you can choose $k = 3$ and proceed as follows:

- choose the substring from the 1-st to the 3-rd character and obtain $s = 11000$;
- choose the substring from the 3-rd to the 5-th character and obtain $s = 11111$;

Find the maximum value of k for which it is possible to make all the characters in the string equal to '1' using the described operations. Note that the number of operations required to achieve this is not important.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 5000$) — the length of the string s.

The second line of each test case contains a string s of length n, consisting of the characters '1' and '0'.

It is guaranteed that the sum of the values n^2 over all test cases in the test does not exceed $25 \cdot 10^6$.

Output

For each test case, output the maximum integer k ($1 \le k \le n$) for which it is possible to obtain a string s consisting only of the characters '1' using the described operations.

Standard Input	Standard Output
5 5 00100 5 01000 7 1011101 3 000 2 10	3 2 4 3 1

,

F. Unfair Game

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

Alice and Bob gathered in the evening to play an exciting game on a sequence of n integers, each integer of the sequence **doesn't exceed** 4. The rules of the game are too complex to describe, so let's just describe the winning condition — Alice wins if the bitwise XOR of all the numbers in the sequence is non-zero; otherwise, Bob wins.

The guys invited Eve to act as a judge. Initially, Alice and Bob play with n numbers. After one game, Eve removes one of the numbers from the sequence, then Alice and Bob play with $n - 1$ numbers. Eve removes one number again, after which Alice and Bob play with $n - 2$ numbers. This continues until the sequence of numbers is empty.

Eve seems to think that in such a game, Alice almost always wins, so she wants Bob to win as many times as possible. Determine the maximum number of times Bob can win against Alice if Eve removes the numbers optimally.

Input

The first line contains an integer t ($1 \leq t \leq 10^4$) — the number of test cases.

The first and only line of each test case contains four integers p_i ($0 \leq p_i \leq 200$) — the number of ones, twos, threes, and fours in the sequence at the beginning of the game.

Output

For each test case, print the maximum number of times Bob will win in a separate line, if Eve removes the numbers optimally.

Standard Input	Standard Output
5 1 1 1 0 1 0 1 2 2 2 2 0 3 3 2 0 0 9 9 9	1 1 3 3 12

Note

In the first example, Bob wins when Eve has not removed any numbers yet.

In the second example, Bob wins if Eve removes one one and one three.

,

G. GCD on a grid

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 256 megabytes

Not long ago, Egor learned about the Euclidean algorithm for finding the greatest common divisor of two numbers. The greatest common divisor of two numbers a and b is the largest number that divides both a and b without leaving a remainder. With this knowledge, Egor can solve a problem that he once couldn't.

Vasily has a grid with n rows and m columns, and the integer a_{ij} is located at the intersection of the i-th row and the j-th column. Egor wants to go from the top left corner (at the intersection of the first row and the first column) to the bottom right corner (at the intersection of the last row and the last column) and find the greatest common divisor of all the numbers along the path. He is only allowed to move down and to the right. Egor has written down several paths and obtained different GCD values. He became interested in finding the maximum possible GCD.

Unfortunately, Egor is tired of calculating GCDs, so he asks for your help in finding the maximum GCD of the integers along the path from the top left corner to the bottom right corner of the grid.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and m ($1 \le n, m \le 100$) — the number of rows and columns of the grid.

Then, there are n lines, where the i-th line contains m integers ($1 \le a_{i,j} \le 10^6$) — the integers written in the i-th row and the j-th column of the grid.

It is guaranteed that the sum of $n \cdot m$ does not exceed $2 \cdot 10^5$ over all test cases.

Output

For each test case, output the maximum possible GCD along the path from the top left cell to the bottom right cell in a separate line.

Standard Input	Standard Output
3 2 3 30 20 30 15 25 40 3 3 12 4 9 3 12 2 8 3 12 2 4 2 4 6 8 1 3 6 9	10 3 1

,

H. The Most Reckless Defense

Input file:	standard input
Output file:	standard output
Time limit:	3 seconds
Memory limit:	256 megabytes

You are playing a very popular Tower Defense game called "Runnerfield 2". In this game, the player sets up defensive towers that attack enemies moving from a certain starting point to the player's base.

You are given a grid of size $n \times m$, on which k towers are already placed and a path is laid out through which enemies will move. The cell at the intersection of the x-th row and the y-th column is denoted as (x, y).

Each second, a tower deals p_i units of damage to all enemies within its range. For example, if an enemy is located at cell (x, y) and a tower is at (x_i, y_i) with a range of r, then the enemy will take damage of p_i if $(x - x_i)^2 + (y - y_i)^2 \leq r^2$.

Enemies move from cell $(1, 1)$ to cell (n, m), visiting each cell of the path exactly once. An enemy instantly moves to an adjacent cell horizontally or vertically, but before doing so, it spends one second in the current cell. If its health becomes zero or less during this second, the enemy can no longer move. The player loses if an enemy reaches cell (n, m) and can make one more move.

By default, all towers have a zero range, but the player can set a tower's range to an integer r ($r > 0$), in which case the health of all enemies will increase by 3^r. However, each r can only be used for **at most one** tower.

Suppose an enemy has a base health of h units. If the tower ranges are 2, 4, and 5, then the enemy's health at the start of the path will be $h + 3^2 + 3^4 + 3^5 = h + 9 + 81 + 243 = h + 333$. The choice of ranges is made once before the appearance of enemies and cannot be changed after the game starts.

Find the maximum amount of base health h for which it is possible to set the ranges so that the player does not lose when an enemy with health h passes through (without considering the additions for tower ranges).

Input

The first line contains an integer t ($1 \leq t \leq 100$) — the number of test cases.

The first line of each test case contains three integers n, m, and k ($2 \leq n, m \leq 50, 1 \leq k < n \cdot m$) — the dimensions of the field and the number of towers on it.

The next n lines each contain m characters — the description of each row of the field, where the character "." denotes an empty cell, and the character "#" denotes a path cell that the enemies will pass through.

Then follow k lines — the description of the towers. Each line of description contains three integers x_i, y_i, and p_i ($1 \leq x_i \leq n, 1 \leq y_i \leq m, 1 \leq p_i \leq 500$) — the coordinates of the tower and its attack parameter. All coordinates correspond to empty cells on the game field, and all pairs (x_i, y_i) are pairwise distinct.

It is guaranteed that the sum of $n \cdot m$ does not exceed 2500 for all test cases.

Output

For each test case, output the maximum amount of base health h on a separate line, for which it is possible to set the ranges so that the player does not lose when an enemy with health h passes through (without considering the additions for tower ranges).

If it is impossible to choose ranges even for an enemy with 1 unit of base health, output "0".

Standard Input	Standard Output
6 2 2 1 #. ## 1 2 1 2 2 1 #. ## 1 2 2 2 2 1 #. ## 1 2 500 3 3 2 #.. ##. .## 1 2 4 3 1 3 3 5 2 #.### #.#.# ###.# 2 2 2 2 4 2 5 5 4 #.... #.... #.... #.... ##### 3 2 142 4 5 9 2 5 79 1 3 50	0 1 1491 11 8 1797

Note

In the first example, there is no point in increasing the tower range, as it will not be able to deal enough damage to the monster even with 1 unit of health.

In the second example, the tower has a range of 1, and it deals damage to the monster in cells $(1, 1)$ and $(2, 2)$.

In the third example, the tower has a range of 2, and it deals damage to the monster in all path cells. If the enemy's base health is 1491, then after the addition for the tower range, its health will be $1491 + 3^2 = 1500$, which exactly equals the damage the tower will deal to it in three seconds.

In the fourth example, the tower at $(1, 2)$ has a range of 1, and the tower at $(3, 1)$ has a range of 2.