Problem A: Paper Cut

Still remember those games we played in our childhood? Folding and cutting paper must be among the most popular ones. Clever children will always search for something new, even when they play games like cutting paper. Now, Carol, a smart girl, asks her brother Mike to solve a puzzle. However, as always, Mike cannot find the solution, therefore he turns to you for help.

Carol's puzzle is simple to state. She folds the paper in a certain manner and then uses a knife to cut through the folded paper. What Mike needs to do is to tell how many pieces the folded paper will turn into after it is cut. To eliminate the ambiguity, we can coordinate the paper as [0, 1] * [0, 1], with the coordinates of lower left corner (0, 0). A fold is denoted by two points (x1, y1) and (x2, y2) on the folding line, with which, the direction of the line is determined by from (x1, y1) to (x2, y2). Carol will always fold the paper from left to right relative to the directed line given (see Figure-1). The cut is determined by the two points on the cut line. Please note that the points given to determine the fold or the cut are not necessarily on the paper.

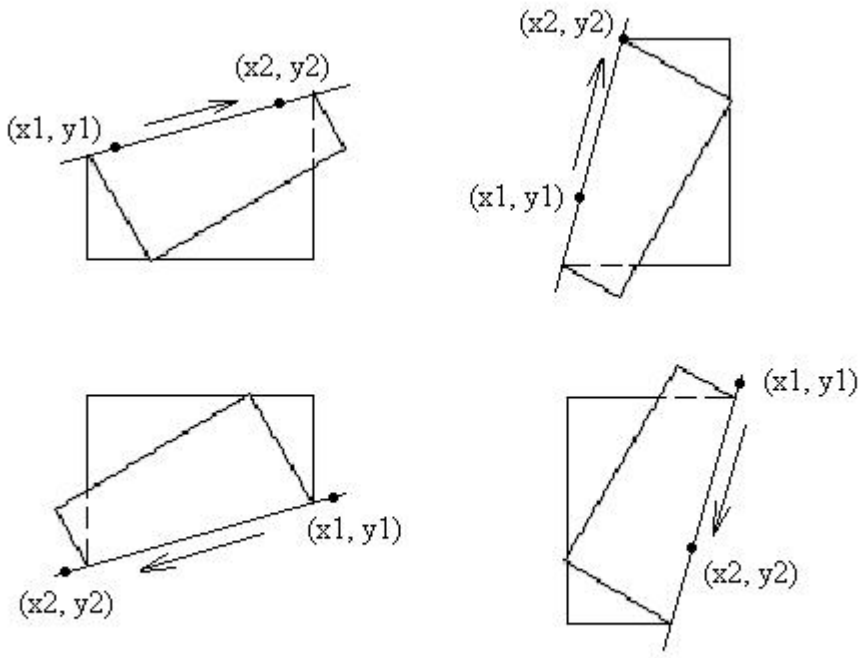


Figure-1 Permissible folding ways

Input

The first line of the input contains one integer t, the number of test cases. Then t cases follow. For each test case, the first line consists of an integer N (0 <= N <= 20), the number of folds, and the

following N lines give two points on each fold line as x1, y1, x2, y2. The following line gives two points on the cut line in the same way.

Output

For each test case, output one line containing the number of pieces the paper will turn into after the cut.

Sample Input

```
2
1
0 0.5 1 1
0.5 0 0.5 1
1
0 0.5 1 1
0 0.4 1 0.4
```

Sample Output

```
2
3
```

Problem B: Ride to School

Many graduate students of Peking University are living in Wanliu Campus, which is 4.5 kilometers from the main campus – Yanyuan. Students in Wanliu have to either take a bus or ride a bike to go to school. Due to the bad traffic in Beijing, many students choose to ride a bike.

We may assume that all the students except "Charley" ride from Wanliu to Yanyuan at a fixed speed. Charley is a student with a different riding habit – he always tries to follow another rider to avoid riding alone. When Charley gets to the gate of Wanliu, he will look for someone who is setting off to Yanyuan. If he finds someone, he will follow that rider, or if not, he will wait for someone to follow. On the way from Wanliu to Yanyuan, at any time if a faster student surpassed Charley, he will leave the rider he is following and speed up to follow the faster one.

We assume the time that Charley gets to the gate of Wanliu is zero. Given the set off time and speed of the other students, your task is to give the time when Charley arrives at Yanyuan.

Input

There are several test cases. The first line of each case is N (1 <= N <= 10000) representing the number of riders (excluding Charley). N = 0 ends the input. The following N lines are information of N different riders, in such format:

Vi [TAB] Ti

Vi is a positive integer <= 40, indicating the speed of the i-th rider (kph, kilometers per hour). Ti is the set off time of the i-th rider, which is an integer and counted in seconds. In any case it is assured that there always exists a nonnegative Ti.

Output

Output one line for each case: the arrival time of Charley. Round up (ceiling) the value when dealing with a fraction.

Sample Input

4

20 0
25 -155
27 190
30 240
2
21 0
22 34
0

Sample Output

780
771

Problem C: Fourier's Lines

Joseph Fourier was a great mathematician and physicist and is well known for his mathematic series. Among all the nineteen children in his family, Joseph was the youngest and the smartest. He began to show his interest in mathematics when he was very young. After he grew up, he often corresponded with C. Bonard (a professor of mathematics at Auxerre) by exchanging letters.

In one letter written to Bonard, Fourier asked a question: how to draw 17 lines on a plane to make exactly 101 crossings, where each crossing belongs to exactly two lines. Obviously, this is an easy problem, and Figure-1 is a solution that satisfies his requirement. Now the problem for you is a universal one. Can we draw N lines on a plane to make exactly M crossings, where each crossing belongs to exactly two lines? If we can, how many pieces, at most, can these lines cut the plane into?

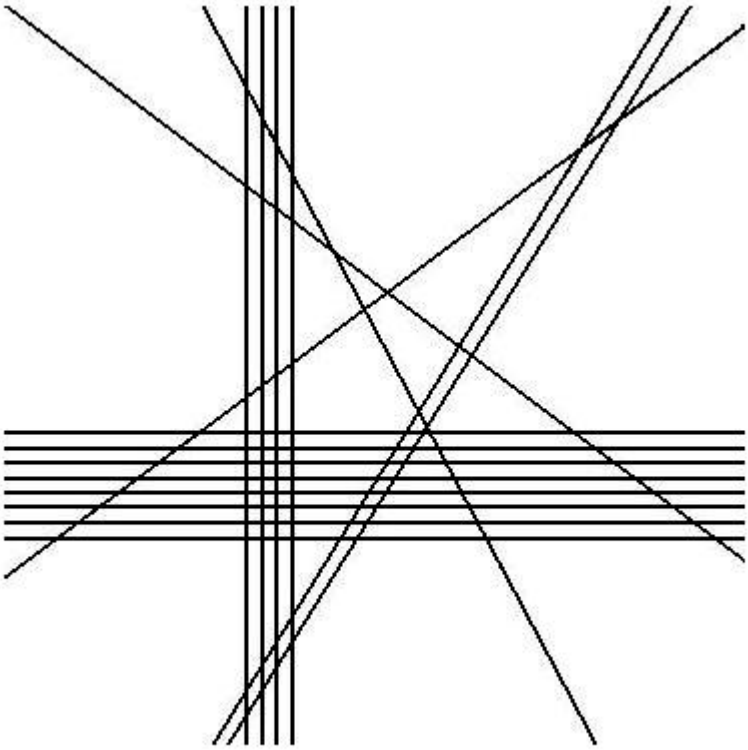


Figure-1 17 lines make exactly 101 crossings

Input

The input may have several sets of test data. Each set is one line containing two integers N and M (1 <= N <= 100, 0 <= M <= 10000), separated by a space. The test data is followed by a line containing two zeros, which indicates the end of input and should not be processed as a set of data.

Output

Output one line for each set of input in the following format:

Case i: N lines cannot make exactly M crossings.

if the drawing of these lines is impossible;

or:

Case i: N lines with exactly M crossings can cut the plane into K pieces at most.

Note: Even if N or M equals to one, you should use the words "lines" and "crossings" in your output.

Sample Input

```
4 3
4 6
4 2
5 11
17 101
0 0
```

Sample Output

```
Case 1: 4 lines with exactly 3 crossings can cut the plane into 8 pieces at most.
Case 2: 4 lines with exactly 6 crossings can cut the plane into 11 pieces at most.
Case 3: 4 lines cannot make exactly 2 crossings.
Case 4: 5 lines cannot make exactly 11 crossings.
Case 5: 17 lines with exactly 101 crossings can cut the plane into 119 pieces at most.
```

Problem D: The Treasure

We have arrived at the age of the Internet. Many software applications have transformed from stand-alone to online applications. Computer games are following this trend as well. Online games are becoming more and more popular, not only because they are more intelligent, but also because they can bring great profits. "The computer game industry is developing rapidly in China. Online game revenues amounted to 1.3 billion Yuan last year and are expected to reach 6.7 billion Yuan by 2007." reported by China Daily in 2004.

However, good games originate from good programmers. We take for example that there is a RPG (Role Playing Game) and your boss asks you to implement some tasks. For simplicity's sake, we assume there are two kinds of roles in this game: one is player and the other is monster. You should help the player to achieve the goal: reach the place where treasure is positioned as early as possible and get the treasure.

The map of the game is a matrix of N * M identical cells. Some cells are passable blocks, and others are non-passable rocks. At any time, there is at most one role occupying a block.
At the beginning, the time is set to 0, and the player is at a certain block. He then moves towards the treasure. At each turn, we have some rules:

- The player can stay in the same block during the next one-second time duration, or he can walk or run towards the east, south, west, north, northeast, northwest, southeast, and southwest.

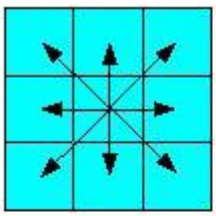


Fig.1 Walk to neighbor blocks

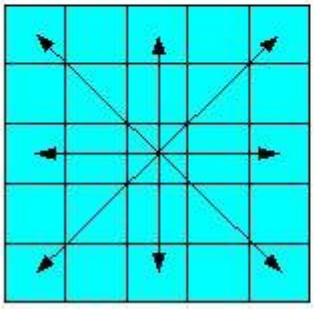


Fig.2 Run at 8 directions

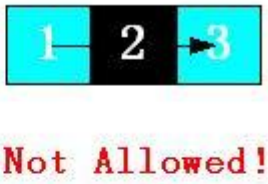


Fig.3 An unallowed running strategy

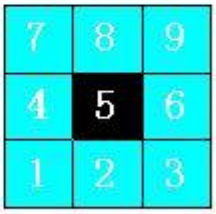


Fig.4 Aggressive monsters

- With walking, the player can arrive at the corresponding passable blocks around him (See Fig.1). Each move takes 1 second.
- With running, the player can arrive at the corresponding passable blocks 2 cells away from him (See Fig.2). Each run takes 1 second. As demonstrated in Fig.3, if a neighbor cell is not passable, the player cannot run in that direction. For example, if cell 2 is a rock, running from 1 to 3 is impossible.
- The monsters are classified into aggressive and non-aggressive. If a monster occupies a cell, the player cannot move into that cell or run through that cell. In addition, the player cannot move into the cells surrounding an aggressive monster, because it will attack the player near it. For example, in Fig.4, if there is an aggressive monster in 5, then the cell 1, 2, 3, 4, 6, 7, 8 and 9 are in its attacking region, so the player cannot stay in or pass through these cells.
- Monsters change their positions each turn. Each monster appears by its position sequence iteratively. That's to say, given the position sequence of monster i: (x1, y1), (x2, y2), ..., (xs, ys), its initial position is (x1, y1) at time 0, then it appears in (x2, y2) at time 1, and so on. When monster i arrives at (xs, ys) at time s-1, it will arrive in (x1, y1) at time s, and start to repeat.
- At the start of each turn, all the monsters change their positions first (the way of changing is given above). If a monster appears in the player's cell, or if an aggressive monster appears near the player to put him in its attacking region, the player will die, and the goal cannot be achieved. After all the monsters change their positions, the player makes a move or stays in the same cell. In his move, the moving path should not be occupied by any rocks or monsters or in the attacking region of any aggressive monsters. When counting the total time, we can neglect the time between monsters' position change and the player's move.

Given the map of the game, the player's starting position, the treasure position and all the monsters' positions in every second, your task is to write a program to find the minimum time that the player gets the treasure.

Input

The input consists of several test cases. The first line of each case contains two integers N and M (1 <= N, M <= 100), where N is the height of the map and M is the width of the map. This is followed by N lines each containing M characters representing the map. A '#' represents a rock, a '.' is a free block, 'p' is the starting position of the player, 't' is the position of the treasure, 'n' is the initial position of a non-aggressive monster, and an 'a' stands for the initial position of an aggressive monster.

The cell (i, j) is the j-th cell on the i-th row counting from left to right. The rows are counted from 1 to N starting from the first line of the matrix. We can number all the monsters as 1, 2, 3… according to their initial position, sorting first by row, then by column.
The (n+2)-th line contains an integer p (0 <= p <= 100), which is the total number of monsters (i.e. the total number of 'n's and 'a's in the matrix). It is followed by p lines each specifying a monster's

position sequence in the following format: the i-th (1 <= i <= p) line corresponds to monster i, which begins with an integer s (1 <= s <= 100), meaning the length of position sequence. Then s pairs of integers x1, y1, x2, y2, …, xs, ys are followed, separated by blanks. Each pair is a free block in the map, (i.e. a monster never goes to a rock cell).

It is assured that none of the aggressive monsters' initial position is around the player. Two consecutive cases are separated by a blank line. The input is terminated by a line containing a pair of zeros.

Output

For each test case, output the minimum total time required for the player to get the treasure, in seconds. If it's not possible to get the treasure, or the minimum required time is greater than 100 seconds, please print a line just containing the string "impossible". Two consecutive cases should be separated by a blank line.

Sample Input

```
7 8
#.#####.
#.t#..p.
#..#....
..#a.#.#
#...##.n
.#......
........
2
2 4 4 5 4
3 5 8 6 8 5 7

3 3
p#.
##.
t..
0

2 2
#t
p#
0

0 0
```

Sample Output

8

impossible

1

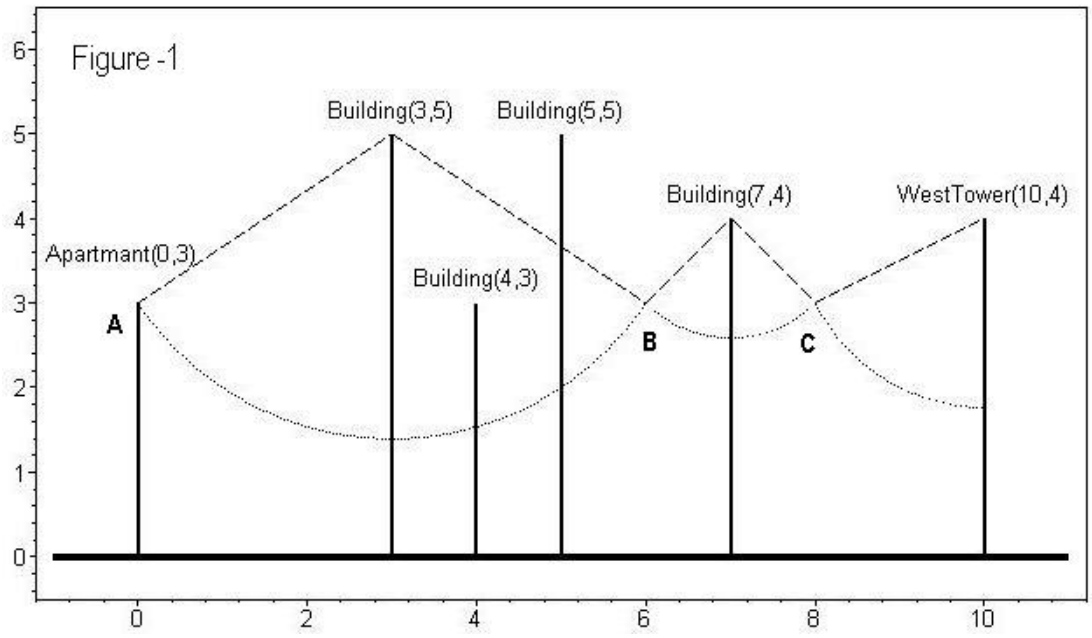
Hint

In the first sample case, the player can follow (2,7), (4,7), stay in (4,7), (6,7), (7,6), (7,4), (5,2), (3,2) and (2,3) to get the treasure with the minimum time (8 seconds).

Problem E: Spiderman

Dr. Octopus kidnapped Spiderman's girlfriend M.J. and kept her in the West Tower. Now the hero, Spiderman, has to reach the tower as soon as he can to rescue her, using his own weapon, the web.

From Spiderman's apartment, where he starts, to the tower there is a straight road. Alongside of the road stand many tall buildings, which are definitely taller or equal to his apartment. Spiderman can shoot his web to the top of any building between the tower and himself (including the tower), and then swing to the other side of the building. At the moment he finishes the swing, he can shoot his web to another building and make another swing until he gets to the west tower. Figure-1 shows how Spiderman gets to the tower from the top of his apartment – he swings from A to B, from B to C, and from C to the tower. All the buildings (including the tower) are treated as straight lines, and during his swings he can't hit the ground, which means the length of the web is shorter or equal to the height of the building. Notice that during Spiderman's swings, he can never go backwards.



You may assume that each swing takes a unit of time. As in Figure-1, Spiderman used 3 swings to reach the tower, and you can easily find out that there is no better way.

Input

The first line of the input contains the number of test cases K (1 <= K <= 20). Each case starts with a line containing a single integer N (2 <= N <= 5000), the number of buildings (including the

apartment and the tower). N lines follow and each line contains two integers Xi, Yi, (0 <= Xi, Yi <= 1000000) the position and height of the building. The first building is always the apartment and the last one is always the tower. The input is sorted by Xi value in ascending order and no two buildings have the same X value.

Output

For each test case, output one line containing the minimum number of swings (if it's possible to reach the tower) or -1 if Spiderman can't reach the tower.

Sample Input

```
2
6
0 3
3 5
4 3
5 5
7 4
10 4
3
0 3
3 4
10 4
```

Sample Output

```
3
-1
```

Problem F: Pollution

The managers of a chemical plant, which is notorious for its high pollution, plan to adopt a newly developed device in order to reduce the amount of contaminants emitted. However, engineers in the plant are against this plan, as they doubt the usefulness of the device. As engineers only believe in experimental results, managers decide to hire programmers to make a numerical experiment to convince the engineers.

The new pollution-reducing device consists of several tanks with pipes connecting the tanks. You may assume there is at most one pipe between two tanks. Two tanks are called adjacent if a pipe connects them. When operating, the contaminant circulates in the device among these tanks.

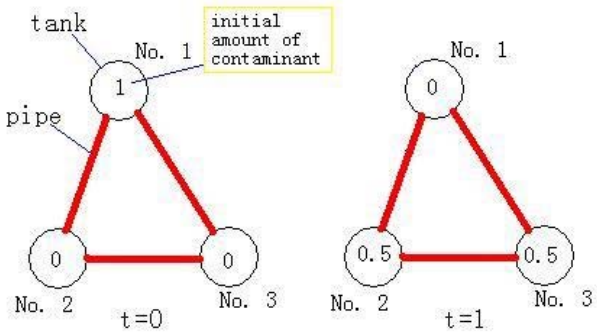


Figure-1

As shown in the Figure-1, the contaminant in one tank in time t, will equally distribute into all adjacent tanks in the time t+1. In other words, if we use X_i^t to denote the amount of contaminant in tank i at time t, we can use the following formula:

$$X_i^{t+1} = \sum_j I_{ij} X_j^t / d_j,$$

where I_{ij}=1 if tank i and tank j are adjacent, otherwise I_{ij}=0, and where d_j is the number of tanks adjacent to tank j. If no tank is adjacent to tank i, we have $X_i^{t+1}=X_i^t$.
The managers, as well as the engineers, want to know that given the initial amount of contaminant in each tank, how the contaminant will be distributed in all the tanks after a long period of time in circulation. Namely, given X_i^0 for all i, what are X_i^t when the difference between X_i^t and X_i^{t+1} is so small that it can be ignored. You may assume that this condition will ALWAYS be attained from an initial case in this problem.

Input

The first line of the input contains one integer T (1 <= T <= 10), the number of test cases. T cases then follow. For each test case, the first line consists of two integers: N and M where(1 <= N <=

100, 0 <= M <= N*(N-1)/2), is the number of tanks and pipes. The following N lines give the initial amount of contaminant for each tank, which are nonnegative real numbers and no larger than 100. Then the next M lines give the tanks that each pipe connects, as "A B" (1 <= A, B <= N, A != B) denotes there is a pipe between tank A and tank B.

Output

For each test case, output the final amount of contaminant X_i^{t+1} (one per line), followed by a blank line. The number should be rounded to three digits after the decimal point.

Sample Input

```
2
3 3
1
0
0
1 2
2 3
3 1
4 4
1
0
0
1
1 2
2 3
3 1
3 4
```

Sample Output

```
0.333
0.333
0.333

0.500
0.500
0.750
0.250
```

Problem G: Area in Triangle

Given a triangle field and a rope of a certain length (Figure-1), you are required to use the rope to enclose a region within the field and make the region as large as possible.

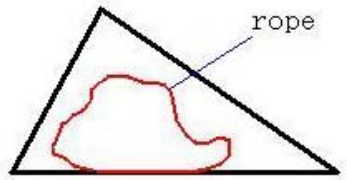


Figure-1

Input

The input has several sets of test data. Each set is one line containing four numbers separated by a space. The first three indicate the lengths of the edges of the triangle field, and the fourth is the length of the rope. Each of the four numbers have exactly four digits after the decimal point. The line containing four zeros ends the input and should not be processed. You can assume each of the edges are not longer than 100.0000 and the length of the rope is not longer than the perimeter of the field.

Output

Output one line for each case in the following format:

Case i: X

Where i is the case number, and X is the largest area which is rounded to two digits after the decimal point.

Sample Input

```
12.0000 23.0000 17.0000 40.0000
84.0000 35.0000 91.0000 210.0000
100.0000 100.0000 100.0000 181.3800
0 0 0 0
```

Sample Output

```
Case 1: 89.35
Case 2: 1470.00
Case 3: 2618.00
```

Problem H: The Peanuts

Mr. Robinson and his pet monkey Dodo love peanuts very much. One day while they were having a walk on a country road, Dodo found a sign by the road, pasted with a small piece of paper, saying "Free Peanuts Here! " You can imagine how happy Mr. Robinson and Dodo were.

There was a peanut field on one side of the road. The peanuts were planted on the intersecting points of a grid as shown in Figure-1. At each point, there are either zero or more peanuts. For example, in Figure-2, only four points have more than zero peanuts, and the numbers are 15, 13, 9 and 7 respectively. One could only walk from an intersection point to one of the four adjacent points, taking one unit of time. It also takes one unit of time to do one of the following: to walk from the road to the field, to walk from the field to the road, or pick peanuts on a point.

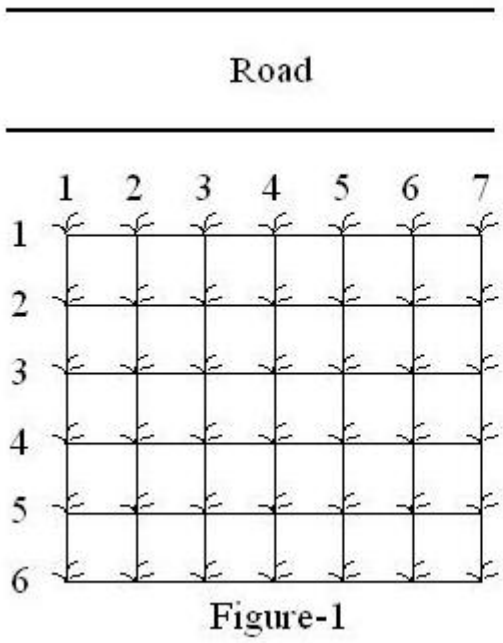


Figure-1

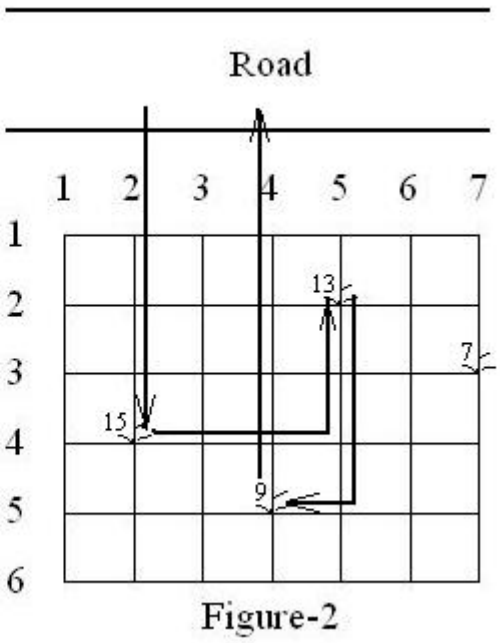


Figure-2

According to Mr. Robinson's requirement, Dodo should go to the plant with the most peanuts first. After picking them, he should then go to the next plant with the most peanuts, and so on. Mr. Robinson was not so patient as to wait for Dodo to pick all the peanuts and he asked Dodo to return to the road in a certain period of time. For example, Dodo could pick 37 peanuts within 21 units of time in the situation given in Figure-2.

Your task is, given the distribution of the peanuts and a certain period of time, tell how many peanuts Dodo could pick. You can assume that each point contains a different amount of peanuts, except 0, which may appear more than once.

Input

The first line of input contains the test case number T (1 <= T <= 20). For each test case, the first line contains three integers, M, N and K (1 <= M, N <= 50, 0 <= K <= 20000). Each of the following M lines contain N integers. None of the integers will exceed 3000. (M * N) describes the peanut field. The j-th integer X in the i-th line means there are X peanuts on the point (i, j). K means Dodo must return to the road in K units of time.

Output

For each test case, print one line containing the amount of peanuts Dodo can pick.

Sample Input

```
2
6 7 21
0 0 0 0 0 0 0
0 0 0 0 13 0 0
0 0 0 0 0 0 7
0 15 0 0 0 0 0
0 0 0 9 0 0 0
0 0 0 0 0 0 0
6 7 20
0 0 0 0 0 0 0
0 0 0 0 13 0 0
0 0 0 0 0 0 7
0 15 0 0 0 0 0
0 0 0 9 0 0 0
0 0 0 0 0 0 0
```

Sample Output

```
37
28
```