ICPC - Programming Contest
FAU Local Summer Contest 2008

June 7, 2008

Problem Overview:

Good luck and have fun!

Problem A
All rights reserved

Programmers of computer games put a lot of effort into preventing illegal copies of their software. In the good old days and sometimes even today, gamers have to enter character codes on startup or during the installation. What a waste of time, for both the programmer and the player. At least the members of the ICPC, the International Crackers of Program Code, got some nice work for their hobby.

When your nostalgic mode has hit you recently, you went into the attic to browse through some old stuff. That was the moment, you've found the box with old floppies and with some kind of copyright protection "wheel". It works as follows: The bottom layer is a square paper with a square array of characters. The upper layer is another square paper with holes at different locations. To get the copyright code, you first have to turn the upper layer into the starting position. Then you have to type all the characters that are visible through the holes. After typing, you turn the upper layer clockwise by 90 degrees and repeat this procedure until the upper layer of the wheel is at the starting position again.

Unfortunately the upper layer is broken in the past and some small parts are missing. You were able to fix this by taping the remaining parts with sticky tape but you don't know whether the still missing parts do or do not included any holes. You remember one attribute: The holes are arranged in such a way that each character on the lower layer is read exactly once during the four decoding steps. Your task is to find the possibilities of fixing the upper layer completely.
Hint: You will find a message in the sample input, if you decode it correctly.

Input

You get the number of test cases (at most 1000) in a single line.
Each test case is specified as follows. The first line holds $n \leq 500$. The $n \times n$ characters of the lower layer follow in n lines that hold n characters each. The upper layer of the wheel is an $n \times n$ mask, given in n lines with n mask positions each. A mask position `=1` indicates a hole, a `0` tells you that there is no hole, and an `X` stands for a missing information (either hole or no hole). Test cases are separated by an empty line.

Output

Per test case, output a single line that gives the number of possibilities to fix the mask in a correct way. This number will always be smaller than 2^{63}. If you don't have to fix anything, because the test case already provided a complete and valid mask already, output the number `0`. If it is impossible to fix the mask or if you have taped your mask incorrectly output `IMPOSSIBLE` instead.
Sample Input and Output are provided on the next page.

Sample Input

```
4
2
CI
PC
11
XX

2
CI
PC
0X
X0

6
-FIESA
REPRSR
GNATFO
STBILC
HALIEP
CM!PRS
000001
100000
101010
000000
011000
000110

4
CE0E
YSNT
TNH!
EJT0
0X00
00X0
0000
0X0X
```

Sample Output

```
IMPOSSIBLE
2
0
1
```

Problem B
Blackjack

Blackjack is a convenient way for making money, not only in real life but also in computer games. I am usually in bad need of lots of money, but I definitely won't work for it. Thus, in my first adventure, I've gambled in the local casino. Maybe you wonder how I could win all that money? Well, first of all, I'm a notorious lounge lizard, and secondly, I know the guys who wrote the computer game featuring me.
When I play blackjack, I get to openly see several cards. In my computer program, the game is implemented as follows:

1. The dealer always deals out from the top of the deck.
2. All dealt cards lay on the table openly.
3. After a round of blackjack has been played, the dealer puts all open cards in a specific order on the bottom of the deck.
4. Then the dealer shuffles all cards.

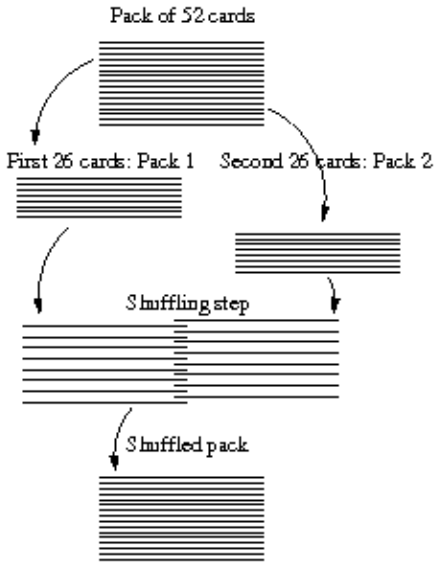

Figure 1 – Shuffle type 1.

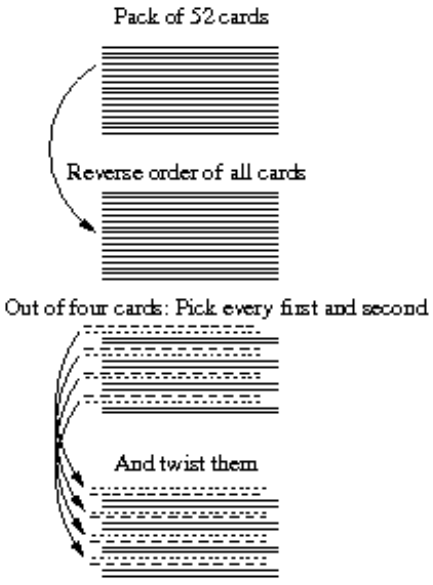

Figure 2 – Shuffle type 2.

The shuffling procedure consists of several phases. Every phase consists of several shuffling steps, and those steps can be of type 1 or type 2. First, all steps of type 1, and then all steps of type 2 are performed in each phase. The number of different shuffling steps does not change between the shuffling phases.

a) Shuffling type 1 (see Fig. 1): Divide the deck (consisting of 52 cards) precisely in two 26-card decks. Then shuffle them in such a way that the bottom-most card is from the first deck, the next card from the second deck, the third card from the first deck, and so on.

b) Shuffling type 2 (see Fig. 2): Reverse the order of the deck and afterwards exchange in every sequence of four cards the first and second card.

5. Start the game again.

Your task is to write a program that tells me if I could know a certain hidden card. Please note that I'm smart enough to also know the 52nd card once I've already seen the 51 other ones :)

Input

The first line of the input consists of a single integer number $c \leq 10000$, the number of test cases, followed by $2 \cdot c$ lines describing the test cases. The first line of a test case contains 5 integer numbers: The number of games $1 \leq g \leq 52$, the number of shuffling phases between two games $1 \leq p \leq 8$, the number of type 1 shuffles $0 \leq t_1 \leq 8$ and the number of type 2 shuffles $0 \leq t_2 \leq 4$. The last integer number n denotes the card you are interested in the $(g+1)$th game (counted from the top of the deck). The second line of a test case contains g integer numbers $s_1, s_2, ..., s_i, ..., s_g$, $1 \leq s_i \leq 10$, stating how many cards have been shown in the ith game. All integers in a line are separated by a single space.

Output

For each test case, now imagine we are in the $g + 1$th game. Print on a single line `I know card number` n`!` (where n is replaced by the actual number n from the input description), if I can predict that card, or `We need to play some more games :(` if not.

Sample Input

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

Sample Output

```
I know card number 1!
We need to play some more games :(
```

Problem C
Building Mountains

Imagine you are a virtual landscaper for computer games. It is your job to design the background for some jump&run game. In your current game project, the protagonist acts in a hilly background. Therefore, for each game level you are given a set of possible heights, where each height occurs exactly once. The heights can be ordered in every possible manner. For the game, the number of ascents in the different levels is important. Your first job is to calculate the number of possible background designs, if you are given a number of heights and a desired number of ascents.
For example, if you have three different heights and these heights must be ordered so that there is one ascent, there are four possibilities (see Fig. 1).

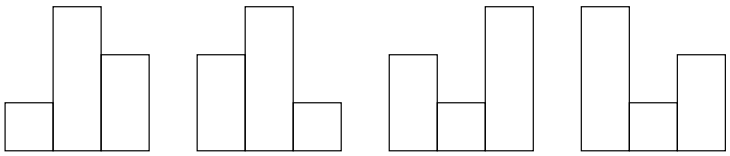

Figure 1 – With numbers these possibilities can be described as 132, 231, 213, 312

Notice that 123 is not contained in the example as it is counted as two ascents instead of one long ascent.

Input

The input starts with a line that holds the number of test cases (at most 30000). Then the test cases follow, each in one line. Each test case consists of two integer numbers separated by a single space. The first integer denotes the number of different heights (at most 100) that must occur in a landscape. The second number denotes the number of desired ascents that must occur in a landscape.

Output

For each test case, calculate the number of possible landscapes. Write each number in a new line.

Sample Input

```
4
20 10
5 2
55 4
3 1
```

Sample Output

```
679562217794156938
66
277483064268921431474435312478017751891
4
```

Problem D
Civ

Sid loves to play strategy games. His favorite one is a simplified version of *Civilization*. In this game, you have to build cities all around the world to claim resources. Each square of the virtual world map has a certain amount of resources that it can produce in each turn of the game. With these resources, Sid can build an army to conquer his enemies. Each city can access the resources of the square it is founded on and of the ones that are directly adjacent to the right and the left. If one square is accessible from more than one city, it can provide its resources only to one of these cities. So each city has access up to a total of three squares of resources.
As building a city is a serious investment, Sid wants to cover as much resources as possible with cities. He usually focuses on the military part of the game and has trouble finding the right spots for his cities. So he asks you to find the best positions for his cities that cover the maximal number of resources on a given map. It is sufficient for him just to know the maximal number of resources for a certain number of cities, so that he knows how many resources he can use to fund his army.

Input

The first line specifies the number of test cases t that follow ($t < 15$). Each test case consists of three numbers $0 < n \leq 15$, $2 < width < 15$, and $2 < height < 15$ on a single line followed by $height$ lines of characters representing the map. The number n denotes the number of cities that should be placed on the map. The map has a size of $height \times width$ squares. Every line of the map has $width$ single digit numbers each of which represent the number of resources available on the respective square. There are no spaces between the numbers. All squares on the border of the map have zero resources.

Output

For each test case, print a single line that gives the maximal number of resources that can be claimed by the given number of cities.
Hint: Sid gets angry, if he has to wait too long – your program should be *quite* fast!

Sample Input

```
2
2 10 10
0000000000
0111111110
0111111110
0111111110
0111111110
0111111110
0111111110
0111111110
0111111110
0000000000
4 3 3
000
090
000
```

Sample Output

```
6
9
```

Problem E
Descent 4

Quite some time after Material Defender 1032 has finished his third mission, the Red Acropolis Team has another job for him. Again, some mines have to be cleared from robots. The first mines were quite easy, but now life's becoming more difficult.

The robots on the Farewell mine can only be destroyed by really heavy bombs. So MD 1032 starts to place one bomb in each room of the mine. However, he has to be extremely careful, because if bombs of the same type explode next to each other, their effect is exponentiated and they will blow away the whole planet.

So he has to use different types of bombs in neighboring rooms, i.e. in rooms that are connected by corridors. If there is no corridor between two rooms, it is safe to place two bombs of the same type because the walls are quite strong. Unfortunately, he has chosen the *Phoenix* ship for his trip to the planet that can only carry very limited amounts of ammunition. Therefore, before he starts to place the bombs, he has to do a quick calculation on how many different bomb types he will need.

Input

There is only one test case per input. The first line of the input contains the number of rooms $n \leq 50$ in the mine, the second line contains the number of corridors $m \leq 200$ between the rooms. Each of the next m lines describes one corridor that exactly connects two rooms: each of these lines contains two integer numbers c_0 and c_1 ($0 \leq c_i < n, c_0 \neq c_1$) separated by a single space. This means that room c_0 and room c_1 are connected through a corridor and may not contain bombs of the same type.

Output

Output the minimum number of different bomb types MD 1032 will need to safely clear all rooms from robots without blowing him and the whole planet away. You may safely assume that no more than 6 different bomb types are needed.

Sample Input

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

Sample Output

```
3
```

Problem F
Game Time

It is party time again at the faculty of mathematics. The party is boring and nobody knows what to do. You suggest to play the all-time number one, favourite mathematician game "Find the number". Unfortunately, there has never been a popular implementation of this game. The game starts with somebody naming an interval and picking a number from the interval. The other players have to find out which number was picked by asking only one type of question: "Is the chosen number smaller than or equal to X?".

The optimal strategy for this game recursively divides the interval in which the number is known to be into two equally sized sub-intervals. Another strategy would be to split the interval into two sub-intervals of different sizes. For example, the smaller sub-interval being a third of the size of the original interval, and the second sub-interval being two thirds of the size of the original interval. Now each strategy needs a certain count of questions to determine the chosen number.

For instance, assume the interval from 1 to 10 (inclusive) and a splitting strategy that cuts the interval into sub-intervals of $\frac{1}{3}$ and $\frac{2}{3}$ of the original size as described above. Somebody else has chosen the number 7. To find this number, you would have to ask the following four questions: Is the number smaller or equal to 4? Is the number smaller or equal to 6? Is the number smaller or equal to 8? Is the number smaller or equal to 7?

For a given strategy, determine the count of questions one needs for each number in the given interval. Sum up the counts for all of these numbers.

Input

The input starts with an integer on a single line, indicating the number of test cases (at most 30). Each following line contains three integers (a, b and c) separated by single spaces. a and b denote the lower and the upper bound of the interval. c denotes the splitting strategy. An interval containing n numbers, $n \geq 2$, is split up according to the following rule:
The first sub-interval contains $\lceil \frac{n}{c} \rceil$ numbers and the second sub-interval contains $\lfloor \frac{nc-n}{c} \rfloor$ numbers.
You may safely assume, that $0 < a \leq b \leq 5 * 10^6$ and $2 \leq c \leq 5 * 10^6$.

Output

Output one line for each test case. Each line must contain the sum of the number of questions which have to be asked for each number in the given interval.

Sample Input

```
5
1 1 2
1 10 2
2 11 2
1 10 3
1 10 5
```

Sample Output

```
0
34
34
34
39
```

Problem G
High Score List

Recently, Ed Logg has been hired by ATARI. As a new programmer, his job is to fix bugs of other programmers. The first project of Ed is on a game called *Asteroids*. The game works great, but there is a problem with the high score list. Sometimes the list is not sorted correctly. ATARI uses old fashioned bubble sort to sort the list due to some obscure reason. And they don't want to change the algorithm. They also don't want Ed to change the code. To circumvent the bug, some of the other programmers wrote the following code:

```
while (isnotsorted(list)) {
      sort(list);
}
```

Ed has to write the function `isnotsorted`. As it is Ed's first day at ATARI, he doesn't want to argue but he doesn't want to help with such an ugly hack either. So he asks you to do it.

Input

The input contains one high score list with exactly ten entries, one per line. Each entry consists of three characters to identify the player, a single space, and the score that the player reached. The score is an integer number and at most 41336440.

Output

Output `0` if the scores are in descending order, `1` else.

Sample Input

```
aaa 100
bbb 99
hw. 90
sc. 90
swe 80
rc. 79
... 0
... 0
... 0
... 0
```

Sample Output

```
0
```

Problem H
Mario Party

Mario Party is a collection of mini games, featuring the famous Italian plumber. For the latest remake, a new mini game is planned that is based on the following idea: The player is on top of a spinning wheel that is divided into several sections of the same size. Mario cannot stand still on the wheel, as he will fall from it while it is spinning. Instead, he has to jump continuously. For each section he touches, points are awarded – but only for the first touch. Hence, the goal is to touch every section exactly once.

As a test player for the ICPC, the Ingenious Computergames Progamer Committee, you are test-playing an early beta of the remake. It seems that your best tactics to be successful is to specialize in jumps of a certain height, and only do jumps of that height. For simplicity, we measure the height of a jump by means of number of sections you pass with one jump. A jump always starts and ends in the middle of a section. So a height of 1 means a jump to the next section. The goal is of course to touch each section with this tactics. For example, a height of 2 sections is not very advisable, if the number of sections is even, as you will only touch half of the sections no matter how long you continue with the game.

Now you wonder, how many different jump heights (between 1 and n) exist that will evenutally touch all n sections of a wheel. Being an efficient player, you will only consider heights that let you touch every section once before touching a single section for a second time. As your profession is gaming and not numeracy, you're already happy knowing the last bit of this number.

Input

The first line of the input starts with the number t ($1 \leq t \leq 50$) of test cases. Then there are t lines, one per test case. Each line provides an integer value n ($2 \leq n < 2^{63}$), the number of sections of the wheel.

Output

For each test case, output a single line that holds the last bit of the desired number, i.e. `0` if the number is even, `1` otherwise.

Sample Input

```
3
2
4
9
```

Sample Output

```
1
0
0
```

Problem I
Restless Lemmings

As child I have been addicted to the famous and funny computer game called *Lemmings*. After school, I had to play several levels of this game in order to get my daily satisfaction. With a new computer and of course a faster processor, the Lemmings on my computer are running much too fast – due to a bug in the program's timing function. I never completed any level since then, because I cannot direct my lemmings fast enough.

But some weeks ago, a friend of mine told me about a flash game that is similar to the original Lemmings but has many more levels. Now, I am addicted again and use every single minute for playing Lemmings.

As I have to sleep, eat, and drink in order to survive, I have to save the game status sometimes and pause. Unfortunately, the pause button is only available **between** levels but not while playing at a level. Since no level codes are given, it is impossible to restart a level if you died. Therefore, I have developed a strategy to deal with that. Fortunately, somewhere on the Internet the average playing time for each level is accounted. From my own experience, I know how long I can play without breaks and how long each break has to be. In my opinion, it must be possible to compute the best playing strategy for me with these information.

Input

The first line of the input gives the number T of test cases ($1 \leq T \leq 10$). Then there follow T test cases, separated from each other by a single empty line. Each test case gives the number of levels L, my maximal playing time (without break) D, and my break time B in its first line ($1 \leq L \leq 100000$; $1 \leq D \leq 1000$; $1 \leq B \leq 60$). L numbers (separated by a single space) follow on the next line that specify the average time for playing these levels (as known from the Internet). The average time for each level is less or equal to D. You may safely assume that each number in the input is an integer.

Output

Print a single line for each test case, containing the shortest time in which I can finish all the levels. You may assume that I always will need exactly the average time to finish a level. Keep in mind that I have to rest not later than after continuously playing for D time units. The break needs to last for at least B time units.
Sample Input and Output are provided on the next page.

Sample Input

```
3
1 100 30
100

3 50 7
13 40 33

3 80 7
60 3 30
```

Sample Output

```
100
100
100
```

Problem J
Tetris

Kenny loves to play puzzle games. His favorite one is an advanced version of Tetris, in which you do not have to combine a sequence of tetrominoes. Instead, you see a given light source and the computer randomly generates points that form a convex polyhedron. As in the original tetris, your goal is to remove the polyhedron. In order to remove that polyhedron, the area of the shadow that results from the projection of the polyhedron to the x-y plane, must be equal to a certain value F. As Kenny is not too good in 3D geometry, he asks you to write a program for him that calculates F.

Hint: The projection of a convex polyhedron always results in a convex polygon shadow.

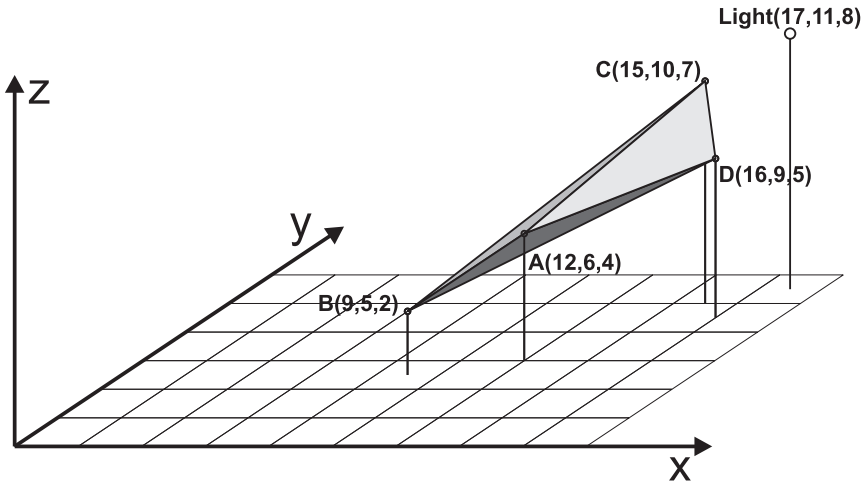

Figure 1 – Polyhedron.

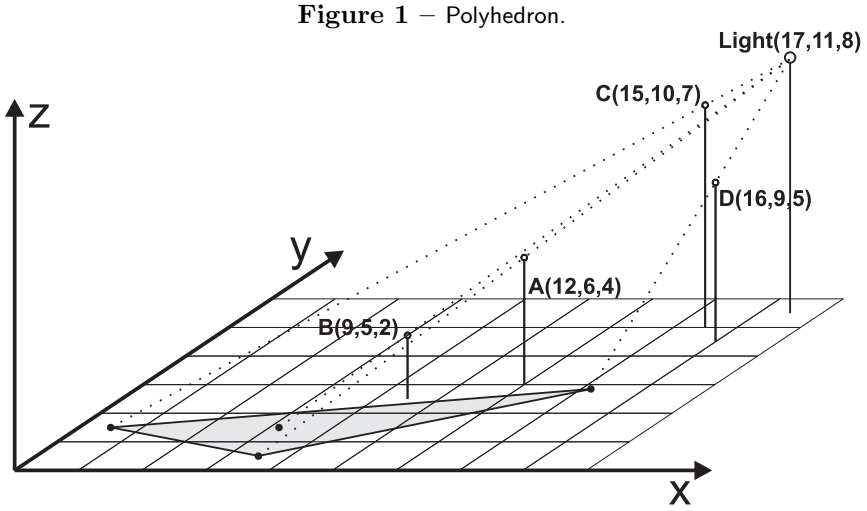

Figure 2 – Projection to x-y-plane.

Input

The first line of the input contains the 3D coordinates of the point where the light source is located. This point and all other points that follow is specified by its x, y and z integer coordinate ($0 \leq$ *coordinate* ≤ 1000). x, y and z coordinates are separated by one space. You may safely assume that each coordinate of the light source is larger than the coordinates of all points in the given polyhedra. The second line holds the number t of test cases/polyhedra ($1 \leq t \leq 100$) in the input. The specification of each polyhedron starts with a line that gives the number of points n ($3 \leq n \leq 500$) that belong to it. Each of the following n lines holds the 3D coordinates of one such point.

Output

For each polyhedron, write the size of the shadow surface area (rounded to 2 decimal places accuracy) on a single line.

Sample Input

```
17 11 8
1
4
12 6 4
13 8 5
15 10 7
16 9 5
```

Sample Output

```
21.33
```