A

A — Ardenia

Welcome to Ardenia. Ardenia is a mythical land, filled with adventure and danger, dwarves and dragons, mages and rouges. And puzzles. Lots of puzzles. Actually, the love for puzzles is the most important life ingredient of the inhabitants, and the only part they have in common.

This month, the people of Ardenia wonder what is the distance between two line segments in three dimensional space. (The distance between segments is defined as the minimum among distances between two points of different segments.) Actually, this problem had originally some motivation, but as nobody from Ardenia cares about motivations, neither should you.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 10^5$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains six space-separated integers $x_1, y_1, z_1, x_2, y_2, z_2 \in [-20, 20]$. Points (x_1, y_1, z_1) and (x_2, y_2, z_2) are the (different) endpoints of the first segment. The second line contains six integers in the same format, describing the second segment.

Single Instance Output

You should output a single line consisting of two co-prime integers ℓ and $m > 0$, such that ℓ/m is the squared distance between the given two segments.

Example

Input	Output
2 0 0 0 1 1 1 1 1 1 2 2 2 1 0 0 0 1 0 1 1 0 2 2 0	0 1 1 2

A

H
Assembly line

The last worker in a production line at the factory of Automated Composed Machinery is worried. She knows that her job hangs in the balance unless her productivity increases. Her work consists of assembling a set of pieces in a given sequence, but the time spent on assembling pieces a and b and then c may not the same as that on assembling pieces b and c, and then assembling a with the resulting component. Only two consecutive pieces may be assembled at a time, and once they are assembled they behave as another piece in terms of the time needed for further assembly.

In order to aid her, you need to find the optimal way to assemble all components. The input to your program will be a set of symbols representing (types of) pieces, and a so-called assembly table representing the time it takes to assemble them, as well as the type of the resulting component. For instance, we may have two symbols $\{a, b\}$, and the following table:

	a	b
a	3–b	5–b
b	6–a	2–b

This means, for example, that two pieces of type a and a may assembled in 3 minutes, and the result is a component of type b, in that the time required to assemble it again with another piece of, say, type a is 6 minutes, and so on. Note that the table is not symmetric, i.e. assembling b and a may be more time-consuming than a and b.

For a sequence of components labelled aba, the two possible solutions are:

- $(ab)a = ba = a$ with time $time(ab) + time(ba) = 5 + 6 = 11$.
- $a(ba) = aa = b$ with time $time(ba) + time(aa) = 6 + 3 = 9$.

So the result for this case would be a piece of type b in 9 minutes (denoted 9–b).

Input

The input consists of several test cases. Each test case begins with a line containing a natural number k ($1 \leq k \leq 26$), followed by a line with k symbols (characters in `[a-z]`) separated by spaces. The following k lines contain the assembly table: the i-th line has k pairs of the form *time*–*result*, where *time* is an integer between 0 and 1 000 000 inclusive, and *result* a symbol belonging to the preceding set. The j-th pair in the i-th line represents the time to compose pieces of types represented by the i-th and j-th symbols, along with the type of the resulting piece. After the table, a line with an integer n indicates the number of lines that follow, each line being a string of at most 200 symbols. Each of these lines is a sequence of components that need to be assembled together in the right order.

The input will finish with a line containing `0`, which should not be processed.

Output

For each test case, print n lines, each with an integer *time* and a symbol *result* in the format *time*–*result*. Each line represents the minimum time and the type of the resulting piece for the corresponding case in the input. In case of a tie among several possible results with the same minimum time, choose from among those the piece whose type letter appears first in the line that contained the k symbols at the beginning of the test case. (For example, if that line was $a\ c\ b$ and both c and b can be obtained with minimum cost 5, print 5–c).

There must be an empty line between the output of different test cases.

Sample Input

```
2
a b
3-b 5-b
6-a 2-b
2
aba
bba
2
m e
5-e 4-m
3-e 4-m
1
eme
0
```

Sample Output

```
9-b
8-a

7-m
```

D Hill Driving

You're driving your car in the local hills and returning to your home town. You'd like to get back as quickly as possible; however, you notice that you don't have much fuel left. You know the most efficient route to take. Some parts of this route go downhill, and some go uphill. The different parts have different lengths and slopes. How quickly can you reach home with the little fuel you have left?

We will assume a very simple model for the fuel consumption of your car. Fuel consumption (per unit distance travelled) will increase linearly with your driving speed v. However, there is an offset which depends on the slope s of the hill. For example, when going downhill along a particular road, you might be able to go at 10 km/h without expending any fuel; on the other hand, if you were travelling that same road uphill, you would expend fuel at the same rate as if you were driving 10 km/h faster along a flat road. More specifically, the car's fuel consumption c in liters per kilometer is given by

$$c = \max(0, \alpha\, v + \beta\, s), \tag{1}$$

where α is the standard fuel consumption rate on a flat road, v is your speed in km/h, s is the slope of the road, and β is a positive constant. Acceleration and deceleration do not cost fuel and can be done instantaneously.

Note that your car has a maximum (safe) speed which cannot be exceeded.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- One line with four floating point numbers α ($0.1 \leq \alpha \leq 100$), β ($0.1 \leq \beta \leq 100$), $v_{\max}$ ($10 \leq v_{\max} \leq 200$) and f ($0 \leq f \leq 50$): the standard (flat road) fuel consumption rate of your car, the slope factor, the maximum speed of your car in km/h, and the amount of fuel you have left in liters, respectively.
- One line with an integer r ($1 \leq r \leq 10\,000$): the number of road segments.
- r lines with two floating point numbers x_i and y_i ($1 \leq x_i \leq 1\,000$, $-1\,000 \leq y \leq 1\,000$) each: the horizontal distance and height change (both in meters) of the i-th road segment. Each road segment has constant slope.

Output

Per test case:

- One line with a floating point number: the fastest time in hours in which you can reach town. It is guaranteed that if it is possible to reach town at all, it will always be possible in less than 24 hours. If it is impossible to reach town, the line must contain "`IMPOSSIBLE`" instead.

Your output should have a relative or absolute error of at most 10^{-6}.

Sample in- and output

Input	Output
3 10.0 1.0 150 0.0 1 100.0 −100.0 10.0 100.0 150 1.0 2 100 0 100 100 0.5 0.1 100 10 3 1000 0 100 10 100 −10	1.414214 IMPOSSIBLE 0.072120

B
Periodic points

Computing the number of fixed points and, more generally, the number of periodic orbits within a dynamical system is a question attracting interest from different fields of research. However, dynamics may turn out to be very complicated to describe, even in seemingly simple models. In this task you will be asked to compute the number of periodic points of period n of a piecewise linear map f mapping the real interval $[0, m]$ into itself. That is to say, given a map $f : [0, m] \to [0, m]$ you have to calculate the number of solutions to the equation $f^n(x) = x$ for $x \in [0, m]$, where f^n is the result of iterating function f a total of n times, i.e.

$$f^n = \overbrace{f \circ .. \circ f \circ f}^{n\ f\text{'s}},$$

where $\circ$ stands for the composition of maps, $(g \circ h)(x) = g(h(x))$.

Fortunately, the maps you will have to work with satisfy some particular properties:

- m will be a positive integer and the image of every integer in $[0, m]$ under f is again an integer in $[0, m]$, that is, for every $k \in \{0, 1, \ldots, m\}$ we have that $f(k) \in \{0, 1, \ldots, m\}$.
- For every $k \in \{0, 1, \ldots, m-1\}$, the map f is linear in the interval $[k, k+1]$. This means that for every $x \in [k, k+1]$, its image satisfies $f(x) = (k + 1 - x)f(k) + (x - k)f(k + 1)$, which is equivalent to its graph on $[k, k + 1]$ being a straight line segment.

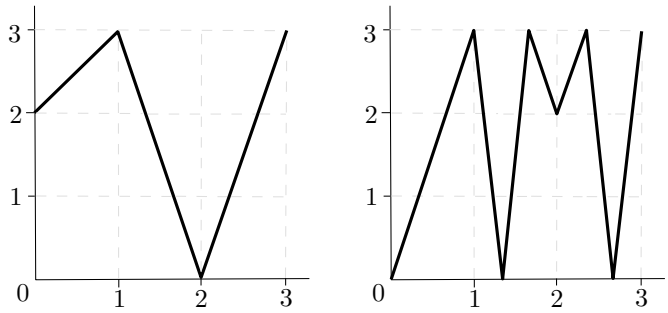


Figure 1: Graphs of the third map in the sample input, f_3 (left), and of its square, f_3^2 (right).

Since there might be many periodic points you will have to output the result modulo an integer.

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing the integer m ($1 \leq m \leq 80$). The following line describes the map f; it contains the $m+1$ integers $f(0), f(1), \ldots, f(m)$, each of them between 0 and m inclusive. The test case ends with a line containing two integers separated by a blank space, n ($1 \leq n \leq 5\,000$) and the modulus used to compute the result, *mod* ($2 \leq mod \leq 10\,000$).

The input will finish with a line containing `0`.

Output

For each case, your program should output the number of solutions to the equation $f^n(x) = x$ in the interval $[0, m]$ modulo *mod*. If there are infinitely many solutions, print `Infinity` instead.

Sample Input

```
2
2 0 2
2 10

3
0 1 3 2
1 137

3
2 3 0 3
20 10000

0
```

Sample Output

```
4
Infinity
9074
```

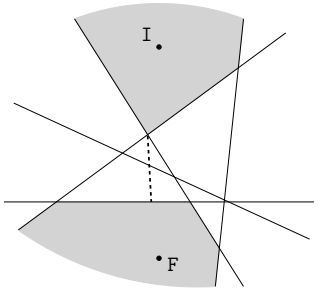
E

B — Beasts

The Fiery Beast and the Ice Beast are relicts of the early days of Ardenia. They live in their lairs and they are kept in the lairs' neighborhoods by powerful magic lines drawn by the Elders. These lines must not be crossed by the beasts. However, in the recent years some of these lines vanished. And if the beasts came too close to each other, the consequences for the whole human race would be catastrophic.

You, the king of the land, are certainly worried about it and gathered all the information about the existing magic lines. It appears that the land can be treated as an infinite plane (this fact was established by Herman the Wise, who traveled for many days in one direction and was still able to see new things!) The lair of the Ice Beast is at point $(0, 10^{10})$ and the lair of the Fiery Beast is at point $(0, -10^{10})$. There are several magic straight infinite lines and neither Fiery nor Ice Beast can cross any of it. You want to know what is the minimum distance which always separates the beasts.

An example is presented below. `I` and `F` denote the beasts' lairs and the gray regions denote the areas on which they may freely walk. Dotted segment corresponds to the minimum distance between these areas.



Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 20$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains one positive integer $n \leq 2 \cdot 10^5$, being the number of magical lines. Each of the next n lines contains three space-separated integers a, b, c, such that $-10^9 \leq a, b, c \leq 10^9$ and $b \neq 0$. These numbers denote the magic line $ax + by + c = 0$.

Single Instance Output

You should output a single line containing one number, being the square of the minimum distance between beasts. Your result is going to be accepted if and only if it is accurate to within a relative or absolute value of at most 10^{-5}.

Example

Input	Output
2 5 1 -1 0 1 1 0 0 1 -6 0 1 -10 0 1 10 4 1 1 10 2 6 7 -1 2 10 0 1 12	400.000000 93.250000

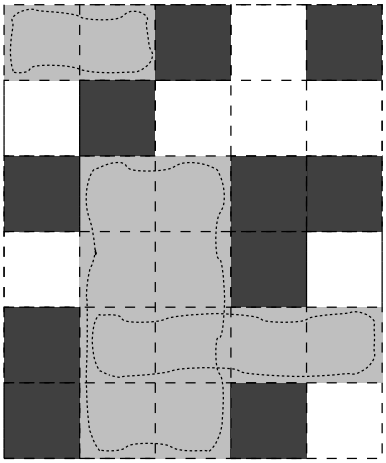
F

G Selling Land

As you may know, the country of Absurdistan is full of abnormalities. For example, the whole country can be divided into unit squares that are either grass or swamp. Also, the country is famous for its incapable bureaucrats. If you want to buy a piece of land (called a parcel), you can only buy a rectangular area, because they cannot handle other shapes. The price of the parcel is determined by them and is proportional to the perimeter of the parcel, since the bureaucrats are unable to multiply integers and thus cannot calculate the area of the parcel.

Per owns a parcel in Absurdistan surrounded by swamp and he wants to sell it, possibly in parts, to some buyers. When he sells a rectangular part of his land, he is obliged to announce this to the local bureaucrats. They will first tell him the price he is supposed to sell it for. Then they will write down the name of the new owner and the coordinates of the south-east corner of the parcel being sold. If somebody else already owns a parcel with a south-east corner at the same spot, the bureaucrats will deny the change of ownership.

Per realizes that he can easily trick the system. He can sell overlapping areas, because bureaucrats only check whether the south-east corners are identical. However, nobody wants to buy a parcel containing swamp.



In this example, dark squares represent swamp. Per may, for example, sell three overlapping grey areas, with dimensions 2×1, 2×4 and 4×1 respectively. The total perimeter is $6 + 12 + 10 = 28$. Note that he can get more money by selling even more land. This figure corresponds to the case in the sample input.

Now Per would like to know how many parcels of each perimeter he needs to sell in order to maximize his profit. Can you help him? You may assume that he can always find a buyer for each piece of land, as long as it doesn't contain any swamps. Also, Per is sure that no square within his parcel is owned by somebody else.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- One line with two integers n and m ($1 \leq n, m \leq 1\,000$): the dimensions of Per's parcel.

F

- n lines, each with m characters. Each character is either '`#`' or '`.`'. The j-th character on the i-th line is a '`#`' if position (i, j) is a swamp, and '`.`' if it is grass. The north-west corner of Per's parcel has coordinates $(1, 1)$, and the south-east corner has coordinates (n, m).

Output

Per test case:

- Zero or more lines containing a complete list of how many parcels of each perimeter Per needs to sell in order to maximize his profit. More specifically, if Per should sell p_i parcels of perimeter i in the optimal solution, output a single line "p_i `x` i". The lines should be sorted in increasing order of i. No two lines should have the same value of i, and you should not output lines with $p_i = 0$.

Sample in- and output

Input	Output
1 6 5 ..#.# .#... #..## ...#. #.... #..#.	6 x 4 5 x 6 5 x 8 3 x 10 1 x 12

D
Fake scoreboard

As you know, after the award ceremony of SWERC it is customary to publish a complete scoreboard with detailed information on the submissions and verdicts received. However, due to the buggy contest management system, most of the relevant data are not being recorded today. Clearly such state of affairs fails to meet the high standards we are committed to, so the judges have resolved to make up the rest of the data based on whatever shred of information left, and hope contestants are unable to tell the difference. To make our lives even simpler, we kindly ask you to provide a solution for us, or else today's scoreboard will remain forever veiled in mystery (even the fake one).

What we will know by the end of the contest is the number T of teams, the number P of problems, and the number of accepted submissions by each team. From the number and colour of balloons floating around on the premises we will also be able to infer how many teams solved each of the problems. Your task is to figure out which teams solved which problems.

Our counting skills are not up to par, so your program should be able to detect when the data we collected must be wrong (see sample input 1). Otherwise you should output a possible solution, represented as a sequence of T strings of P characters each, in the following way. Both problems and teams are assigned with distinct integers, from 1 to P and 1 to T, respectively. For team number i ($1 \leq i \leq T$), write the string on alphabet `N,Y` such that its j-th ($1 \leq j \leq P$) character is `Y` if the team managed to get problem j accepted, and `N` otherwise. For example, the following three strings form a solution to the second sample case, where the score of each of three teams is $2, 1, 2$, and the count of accepted submissions for each of three problems is $1, 2, 2$:

```
NYY
NNY
YYN
```

There is at least one other solution, namely

```
NYY
NYN
YNY
```

When several solutions are possible we ask you to supply the one giving rise to the lexicographically smallest string, when each of the T rows are concatenated in order. In the example above we prefer the first solution, since `NYYNNYYYN` comes before `NYYNYNYNY` in lexicographical order. (String S comes before S' in lexicographical order if the first different character between the two is `N` in S but `Y` in S').

Input

Each input case is described by three lines:

The first contains two space-separated integers T (the number of teams) and P (the number of problems), with $1 \leq T, P \leq 80$. The second contains T space-separated integers between 0 and 90 (inclusive), the i-th of which indicates the number of problems solved by team i. The third (and last) line has P integers between 0 and 90, the j-th of which describes the number of teams successfully solving problem j.

Different input cases are separated by a blank line. The last line of the input file will be `0 0`.

Output

If the input data has a solution, print T lines of P characters each, depicting the lexicographically smallest solution as explained above. Otherwise output a single line with the word `Impossible`. In any case a blank line should separate outputs for different test cases.

Sample Input

```
2 2
1 2
1 1

3 3
2 1 2
1 2 2

3 5
3 3 1
3 1 1 0 2

0 0
```

Sample Output

```
Impossible

NYY
NNY
YYN

YNYNY
YYNNY
YNNNN
```

H

H — Hanging Hats

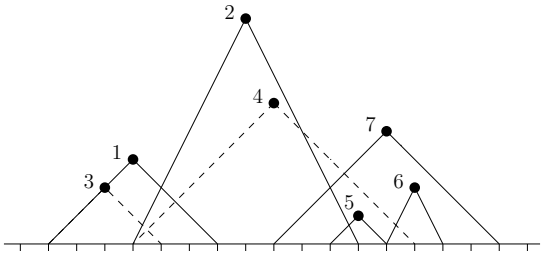
It is a well-known fact that all mages wear pointed hats. However, contrary to popular belief, mages do not sleep in them, and those of the Unheard University hang them on one common, very large wall before going to bed. Mages' hats come only in two shapes: a *wide* one and a *narrow* one.

The mages go to sleep one after another (it is easily assured as due to a tight budget, they have only one bathroom and one towel). Before going to bed, the i-th mage takes its hat and chooses an arbitrary position (x_i, y_i) on the wall, where y_i is the (positive) distance from the ground. Then he hammers a nail at position (x_i, y_i) and hangs his hat on this nail.

Not surprisingly, the hats are magical, which means that they magically unfold to a given height. Hence, when hung, a hat looks exactly like a isosceles triangle of height y_i, i.e., its left and right edges are of equal length. Its top vertex is exactly at the nail and its bottom edge touches the floor. If the hat is narrow, then the length of its bottom edge is y_i; if it is wide, it is equal to $2y_i$.

All nails are provided by the university and they have nice heads which glow in the dark. If a nail becomes covered by a hat hung later (even by its boundary), its glow is no longer visible. Moreover, if a mage tries to hammer a nail at an already hanging hat (also even at its boundary), this nail and his hat are thrown away, and he himself becomes expelled from the university. The Chancellor of the University (apparently not having more important things to worry about) wants to know how the number of visible glowing nail heads changes in time.

An example consisting of 7 hats is presented below. Dots correspond to glowing nail heads; numbers represent the order in which they are hung. Hats 2 and 6 are narrow, the remaining ones are wide. Mages 3 and 4 were expelled while trying to hang their hats (their hats were not hung). After mage 7 hung his hat, hats of mages 1, 2 and 7 were visible.



Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 30$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains $n \in [1, 10^5]$, the number of mages. The i-th of the following n lines contains two space-separated integers $x_i \in [-10^9, 10^9]$ and $y_i \in [1, 10^9]$, followed by a letter `W` or `N`. These numbers are the coordinates where the i-th mage tries to hammer a nail; the letter denotes whether his hat is wide or narrow.

Single Instance Output

For each input instance you should output n lines. The i-th line should contain the word `FAIL` if by hammering the nail the i-th mage got himself expelled from the university. Otherwise, it should contain a positive integer, the number of nail heads visible after the i-th mage hangs his hat.

Example

Input	Output
2 3 0 1 W 0 2 N 0 1 W 7 4 3 W 8 8 N 3 2 W 9 5 W 12 1 W 14 2 N 13 4 W	1 1 FAIL 1 2 FAIL FAIL 3 4 3

E
Palindromic DNA

A DNA sequence is composed of a series of four possible nucleobases, namely Adenine, Guanine, Thymine and Cytosine; we will refer to each of these bases by their initial. For our purposes, nucleobases have an associated cyclic "order": `A` is followed by `G`, which in turn is followed by `T`, which is followed by `C`, which is followed by `A` again. State-of-the-art research in genomics has revealed the startling fact that many diseases are caused by certain subsequences of bases not forming a palindromic sequence! Your mission as a leading researcher at ICPC laboratories is to take a DNA string S and a series of subsets $P_1, \ldots, P_t$ of indices to characters (nucleobases) in S, and transform S so that each of the restrictions of the resulting string to $P_1, \ldots, P_t$ are palindromic. (The restriction of S to a subset $P = \{i_1, i_2, \ldots, i_k\}$ of indices, where $0 \leq i_1 < i_2 < \ldots < i_k < |S|$, is the string $S_{i_1} S_{i_2} \ldots S_{i_k}$). It is possible to inspect any base of S at will, but only three transformations can be applied to a base:

1. Leave it unaltered.
2. Increase it by 1 in the cyclic order of nucleobases (e.g. turn `C` into `A`).
3. Decrease it by 1 (e.g. turn `T` into `G`).

Moreover, owing to limitations of current technology, it is impossible to modify two bases in consecutive positions of the sequence. Is our goal achievable?

By way of example, consider DNA sequence `AGTAT`. Number positions starting from 0, and suppose we have the three subsets $P_1 = \{1, 4\}$, $P_2 = \{0, 1\}$ and $P_3 = \{0, 2, 4\}$. One solution is to increase the first character and decrease the last, yielding $S' =$ `GGTAG`. The restrictions of S' to P_1, P_2 and P_3 are `GG`, `GG` and `GTG`, respectively; all of them are palindromic.

One case where no solution is possible is when the string is `CATGC`, and we require the subsequences determined by positions $\{0, 3\}$ and $\{3, 4\}$ be palindromic. Here, characters 3, 0 and 4 would all need to become a `T`. But this entails modifying consecutive characters 3 and 4, which is not allowed.

Input

The first line of each test case has two integers N and T ($1 \leq N \leq 10\,000, 1 \leq T \leq 6\,000$), the sequence length and number of subsets to consider. The next line contains the DNA sequence of length N, all of whose characters are in `ACGT`. The subsets are described by the following T lines. Each line starts by "L:", where L ($0 \leq L \leq N$) is the number of positions in the subset, and is followed by T distinct integers between 0 and $N - 1$ in increasing order. Subsets may overlap partially or totally.

A blank line separates different test cases. The input file is terminated by a line containing `0 0`.

Output

In a single line per test case, print `YES` if the task is solvable and `NO` otherwise.

Sample Input

```
5 3
AGTAT
2: 1 4
2: 0 1
3: 0 2 4

5 3
CATGC
0:
2: 0 3
2: 3 4

0 0
```

Sample Output

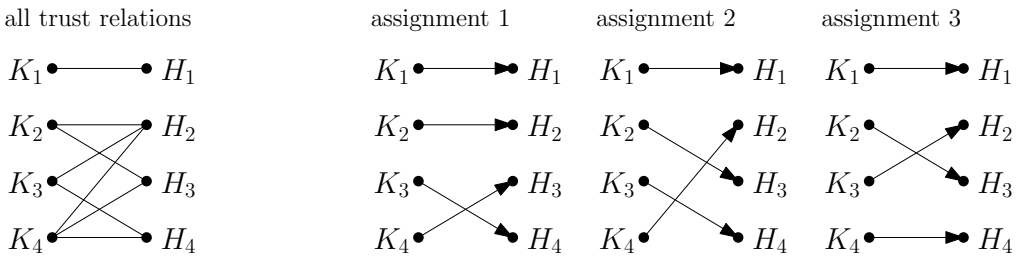
```
YES
NO
```

J

J — Justice for All

Ardenia is going to war! The famous and prestigious Ardenia's military unit consisting of 200 knights and 200 horses started to prepare for battles. During the preparation, k knights and k horses are chosen (the remaining ones simply stay at their barracks) and a special kind of mutual *trust* relation is established between certain knights and certain horses. In such case, we simply say that knight A *trusts* horse B (and vice versa). One horse can trust arbitrarily many knights and one knight can trust arbitrarily many horses.

For a given team of k knights and k horses, the trust relations between them determines the number of battles they are willing to fight. For each battle, each knight chooses a single horse he or she trusts: this creates an *assignment*. An example of 4 knights (K_1, K_2, K_3 and K_4) and 4 horses (H_1, H_2, H_3 and H_4) with trust relations is depicted below. There are 3 possible different assignments.



The assignments for two different battles have to be different (the knights would get bored otherwise) and all possible assignments have to be tried out (the knights are curious enough to test them all). They are pretty good at their fighting skills, which means that no knight or horse will be harmed during the making of the war.

Your mages predicted that the war would consist of n battles. Your task is to choose k knights and k horses and establish trust relations between them, so that they are prepared for exactly n battles.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 100$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The input instance is one line containing the number of battles, $n \in [1, 10^6]$.

Single Instance Output

The first line of the output should contain a single positive integer $k \leq 200$. Each of the next k lines should consist of k binary digits (`0` or `1`, without spaces between them), where `1` in the j-th column of the i-th line means that knight i trusts horse j (and vice versa). The number of possible battles these knights and horses are prepared for should be exactly n.

Example

Input	Output
3 1 3 4	1 1 4 1000 0110 0101 0111 4 1100 1100 0011 0011

G
Sensor network

In response to a request by the SWERC 2010 problem-setting team, a sensor network has just been installed in the headquarters. The organizing committee has put in the problem-setters the disproportionate fear of suffering a leak of classified information about the problems.

Nevertheless, in the rush they forgot to think about the electricity network needed to make the sensors work. Because of this, the security system is currently not working, but we need to justify the great amount of resources invested in it, so the installation must be ready before the end of the contest. Hence you are now asked to elaborate a program which will help the electrician in his duty.

Since the organizing committee spared no expense, they ordered the sensors from a prestigious company. Every sensor is handcrafted and a number is written on each of them, whose meaning is the recommended voltage that should be applied to it for correct operation. They can be used under higher voltages, with an increasing risk of failure, but never with a voltage below the recommended one. The electrician gazed open-mouthed at the sensors when they were given to him: nearly all of them had different recommended voltages! The sensors were installed in the building in such a way that each of them controls exactly two doors and every door is controlled by at least one sensor. Now is the time for the electrician to supply power to the sensors. He faces the following constraints:

- Fortunately, there is no need to activate all sensors. However, we will require that the subset of sensors chosen to be on satisfy that every door is controlled by, at least, one sensor in the subset.
- Electricity is to be supplied to one of the active sensors, and then distributed to the others with wires. In order not to waste wires, they can only be installed by connecting a pair of neighboring active sensors (by "neighbouring" we mean that some door is controlled by both of them). Since we must supply energy to every active sensor, not every subset of sensors is suitable as the chosen subset of working sensors.
- Because of the above, all of the sensors will be supplied the same voltage, which cannot be below the corresponding recommended voltages.

For simplicity, we will refer to a subset of sensors satisfying the first two constraints above as an admissible subset. The network is designed so that the set of all sensors is always admissible, but we would like to take a possibly smaller subset so as to minimize the *margin*, defined as the maximum of the differences, in absolute value, between the numbers written on each pair of sensors in the subset. (This is to keep the chances of failure to a minimum).

The electrician could not solve the task of finding the admissible subset of the set of sensors for which the margin is minimum. Therefore, the electrical installation is still missing. Today, we ask you to write a program to find out the answer, given a sensor network and the recommended voltage for each of the sensors.

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing the integer n ($2 \leq n \leq 350$), the number of doors in the building. The following line contains another integer, m ($n-1 \leq m \leq n(n-1)/2$), the number of sensors in the network. The test case finishes with m lines containing a description of each of the m sensors. The i-th of those lines contains three integers, a ($0 \leq a \leq n-1$), b ($0 \leq b \leq n-1$) and w ($1 \leq w \leq 2^{15}$), in that order. Integers a and b represent the pair of doors controlled by the i-th sensor, and w its recommended voltage. You can safely assume that there are no two sensors controlling the same two doors.

The input will finish with a line containing `0`.

Output

For each case, your program should output a line containing the minimum margin of an admissible subset of the sensors.

Sample Input

```
3
3
0 1 220
1 2 120
2 0 160

4
5
2 3 80
1 3 80
0 1 180
2 1 200
3 0 140

0
```

Sample Output

```
40
60
```