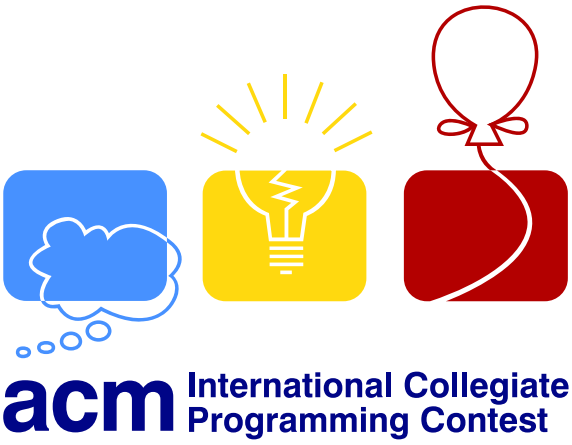
NWERC 2010

The 2010 ACM Northwestern Europe Programming Contest



IBM event sponsor

The Problem Set

A	Fair Division
B	Free Goodies
C	High Score
D	Hill Driving
E	Rankings
F	Risk
G	Selling Land
H	Stock Prices
I	Telephone Network
J	Wormly

Almost blank page

A Fair Division

It's your friend's birthday, and you and some other people decided to buy him a copy of StarCraft II, because who wouldn't want to have that?

You agreed to divide the costs as fairly as possible. Since some of you have more money available than others, you also agreed that nobody has to pay more than he can afford. Every contribution will be a multiple of 1 cent, i.e., nobody can pay fractions of a cent.

Everybody writes down the maximum amount he is able to contribute. Taking into account these maximum amounts from everybody, you share the cost of the present as fairly as possible. That means, you minimize the largest distance of the contributions to $\frac{1}{n}$-th of the total cost. In case of a tie, minimize the second largest distance, and so on. Since the smallest unit of contribution is 1 cent, there might be more than one possible division of the cost. In that case, persons with a higher maximum amount pay more. If there is still ambiguity, those who come first in the list pay more.

Since you bought the present, it is your task to figure out how much everybody has to pay (including you).

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 p and n: the price of the present in cents $(1 \leq p \leq 1\,000\,000)$ and the number of people $(2 \leq n \leq 100)$ who contribute to the present (including you).
- One line with n integers a_i $(1 \leq a_i \leq 1\,000\,000)$, where a_i is the maximum amount, in cents, that the i-th person on the list is able to contribute.

Output

Per test case:

- One line with n integers: the amounts each person has to contribute according to the scheme. If the total cost cannot be divided according to the above rules, the line must contain "`IMPOSSIBLE`" instead.

Sample in- and output

Input	Output
3 20 4 10 10 4 4 7 3 1 1 4 34 5 9 8 9 9 4	6 6 4 4 IMPOSSIBLE 8 7 8 7 4

Almost blank page

B Free Goodies

Petra and Jan have just received a box full of free goodies, and want to divide the goodies between them. However, it is not easy to do this fairly, since they both value different goodies differently.

To divide the goodies, they have decided upon the following procedure: they choose goodies one by one, in turn, until all the goodies are chosen. A coin is tossed to decide who gets to choose the first goodie.

Petra and Jan have different strategies in deciding what to choose. When faced with a choice, Petra always selects the goodie that is most valuable to her. In case of a tie, she is very considerate and picks the one that is least valuable to Jan. (Since Petra and Jan are good friends, they know exactly how much value the other places on each goodie.)

Jan's strategy, however, consists of maximizing his own final value. He is also very considerate, so if multiple choices lead to the same optimal result, he prefers Petra to have as much final value as possible.

You are given the result of the initial coin toss. After Jan and Petra have finished dividing all the goodies between themselves, what is the total value of the goodies each of them ends up with?

Input

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

- One line with an integer n ($1 \leq n \leq 1\,000$): the number of goodies.
- One line with a string, either "`Petra`" or "`Jan`": the person that chooses first.
- n lines with two integers p_i and j_i ($0 \leq p_i, j_i \leq 1\,000$) each: the values that Petra and Jan assign to the i-th goodie, respectively.

Output

Per test case:

- One line with two integers: the value Petra gets and the value Jan gets. Both values must be according to their own valuations.

Sample in- and output

Input	Output
3 4 Petra 100 80 70 80 50 80 30 50 4 Petra 10 1 1 10 6 6 4 4 7 Jan 4 1 3 1 2 1 1 1 1 2 1 3 1 4	170 130 14 16 9 10

C High Score

You've just been playing a video game in which you had to move a worm through a maze using a joystick. You got the high score, and now you have to enter your name using this joystick. This works as follows.

The initial name displayed on the screen is a string consisting only of 'A' characters. Initially the first letter of the string is selected. When you move the joystick forward, the selected letter is changed to the letter that immediately follows it in the alphabet. When you move the joystick backward, the selected letter is changed to the letter that immediately precedes it in the alphabet. The alphabet wraps around, so the letter following 'Z' is 'A' and the letter preceding 'A' is 'Z'.

Moving the joystick left or right changes the selection one step to the left or right, respectively. The selection also wraps around, so moving left when the first letter is selected will select the last letter and vice versa.

Because you would like to spend as little time as possible on entering your name, you want to know the smallest possible number of joystick moves needed to do this. Given the name you want to enter, write a program that calculates the minimum number of moves needed. You may assume that the length of the initial string is the same as the length of the name that you want to enter. Furthermore, it does not matter which letter is selected at the end of the process.

Input

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

- One line with a string s ($1 \leq \text{length}(s) \leq 1\,000$) consisting of uppercase letters: the name that you want to enter.

Output

Per test case:

- One line with an integer: the minimum number of joystick moves needed.

Sample in- and output

Input	Output
2 JEROEN JAN	56 23

Almost blank page

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

E Rankings

There are n teams (labelled from 1 to n) who take part in a programming competition every year, and at the end they are ranked in order of merit. The rankings for last year are known. This year, the jury wants to make the event less competitive, and decides not to publish such a ranking list (since teams near the bottom might get disheartened). Instead, they will produce a complete list of pairs of teams whose relative rank order has changed from last year to this year. For example, if team 13 placed above team 6 last year, but team 6 placed above team 13 this year, the pair (6, 13) is announced. This would enable teams to track their progress against a particular opposing team, but not give them a clear sense of where they stand overall.

Of course, this isn't going to stop your team from trying to determine the overall ranking list. Given last year's rankings and a complete list of the pairs of teams whose relative rank order has changed, reconstruct as much of this year's standings as possible. It is possible that the jury might have made an error, so if the data given is inconsistent with any possible ranking list for this year, you should also detect this.

Input

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

- One line with an integer n ($2 \leq n \leq 500$): the number of teams.
- One line with n integers t_i ($1 \leq t_i \leq n$): the rankings for last year, from best team to worst team. t_i represents the team who came in position i (1-indexed) on the ranklist. All the t_i will be distinct.
- One line with an integer m ($0 \leq m \leq 25\,000$): the number of pairs whose relative rank order has changed.
- m lines with two integers a_i and b_i ($1 \leq a_i < b_i \leq n$) each: a pair of teams whose relative rank order has changed. Each such pair will be mentioned exactly once.

Output

Per test case:

- One line with n integers: the rankings for this year, from best to worst, where the i-th term (1-indexed) represents the team in position i. If this team cannot be determined with certainty, the integer should be replaced with a '`?`' character. If the data for a particular test case is inconsistent with any possible ranking list for this year, the line must contain "`IMPOSSIBLE`" instead.

Sample in- and output

Input	Output
3 5 5 4 3 2 1 2 2 4 3 4 3 2 3 1 0 4 1 2 3 4 3 1 2 3 4 2 3	5 3 2 4 1 2 3 1 IMPOSSIBLE

F Risk

Risk is a board game played on a world map. This world is divided into regions by borders. Each region is controlled by a player (either you or one of your opponents). Any region that you control contains a positive number of your armies.

In each turn, you are allowed to move your armies. Each of your armies may either stay where it is, or move from a region to a bordering region under your control. The movements are performed one by one, in an order of your choice. At all times, each region must contain at least one army.

For strategic purposes, it is important to move your armies to regions that border your opponents' regions and minimize the number of armies on your regions that are totally surrounded by other regions under your control. You want your weakest link, i.e., the border region with the minimum number of armies, to be as strong as possible. What is the maximum number of armies that can be placed on it after one turn?

Input

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

- One line with an integer n ($1 \leq n \leq 100$): the number of regions.
- One line with n integers a_i ($0 \leq a_i \leq 100$): the number of your armies on each region. A number 0 indicates that a region is controlled by your opponents, while a positive number indicates that it is under your control.
- n lines with n characters, where each character is either '`Y`' or '`N`'. The i-th character of the j-th line is '`Y`' if regions i and j border, and '`N`' otherwise. This relationship is symmetric and the i-th character of the i-th line will always be '`N`'.

In every test case, you control at least one region, and your opponents control at least one region. Furthermore, at least one of your regions borders at least one of your opponents' regions.

Output

Per test case:

- One line with an integer: the maximum number of armies on your weakest border region after one turn of moving.

Sample in- and output

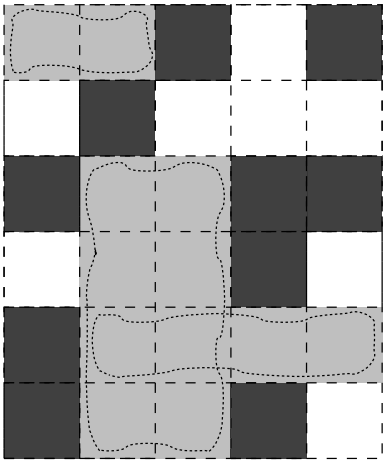
Input	Output
2 3 1 1 0 NYN YNY NYN 7 7 3 3 2 0 0 5 NYNNNNN YNYYNNN NYNYYNN NYYNYNN NNYYNNN NNNNNNY NNNNNYN	1 4

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.

- 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

H Stock Prices

In this problem we deal with the calculation of stock prices. You need to know the following things about stock prices:

- The *ask price* is the lowest price at which someone is willing to sell a share of a stock.
- The *bid price* is the highest price at which someone is willing to buy a share of a stock.
- The *stock price* is the price at which the last deal was established.

Whenever the bid price is greater than or equal to the ask price, a deal is established. A buy order offering the bid price is matched with a sell order demanding the ask price, and shares are exchanged at the rate of the ask price until either the sell order or the buy order (or both) is fulfilled (i.e., the buyer wants no more stocks, or the seller wants to sell no more stocks). You will be given a list of orders (either buy or sell) and you have to calculate, after each order, the current ask price, bid price and stock price.

Input

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

- One line with an integer n ($1 \leq n \leq 1\,000$): the number of orders.
- n lines of the form "*order_type* x `shares at` y", where *order_type* is either "`buy`" or "`sell`", x ($1 \leq x \leq 1\,000$) is the number of shares of a stock someone wishes to buy or to sell, and y ($1 \leq y \leq 1\,000$) is the desired price.

Output

Per test case:

- n lines, each of the form "a_i b_i s_i", where a_i, b_i and s_i are the current ask, bid and stock prices, respectively, after the i-th order has been processed and all possible deals have taken place. Whenever a price is not defined, output "$-$" instead of the price.

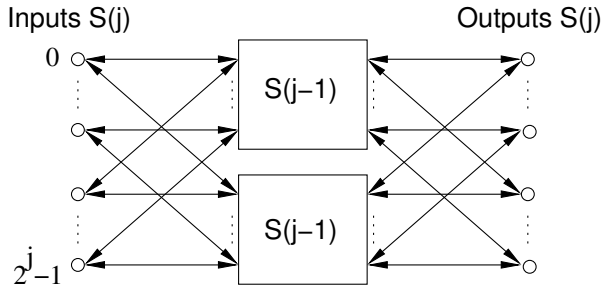
Sample in- and output

Input	Output
2 6 buy 10 shares at 100 sell 1 shares at 120 sell 20 shares at 110 buy 30 shares at 110 sell 10 shares at 99 buy 1 shares at 120 6 sell 10 shares at 100 buy 1 shares at 80 buy 20 shares at 90 sell 30 shares at 90 buy 10 shares at 101 sell 1 shares at 80	- 100 - 120 100 - 110 100 - 120 110 110 120 100 99 - 100 120 100 - - 100 80 - 100 90 - 90 80 90 100 80 90 100 - 80

I Telephone Network

A telephone company wants to build a new telephone network in a city. The company has the goal that each person in the city should be able to call each other person. Of course, it is not possible to build direct connections between every pair of persons. Instead, the company uses a network made up of several layers.

We denote a network switch in layer j by $S(j)$. A switch $S(0)$ consists of one input, one output and a cable connecting the input to the output. A switch $S(j)$ with $j > 0$ consists of 2^j inputs, 2^j outputs and two switches $S(j-1)$. Input i of $S(j)$ ($0 \leq i < 2^j$) is connected via a cable to the inputs i mod 2^{j-1} of each of the two switches $S(j-1)$. Similarly, output i of $S(j)$ is connected to the outputs i mod 2^{j-1} of each of the two switches $S(j-1)$.



The connections between a switch $S(j)$ and the two switches $S(j-1)$ it consists of.

We are considering a network with one switch $S(n)$ in the outermost layer. It can be shown that any input and output of switch $S(n)$ has a unique connection path to any of the $S(0)$ switches. Therefore, any input of $S(n)$ can be connected to any of its outputs, and the connection path is uniquely determined by specifying through which switch $S(0)$ the connection is established.

We number the switches $S(0)$ belonging to the switch $S(n)$ from 0 to $2^n - 1$. The i-th switch $S(0)$ is defined as follows. Write the number i in binary as $b_{n-1}b_{n-2}\ldots b_0$. This defines a path from an input of $S(n)$ to the i-th switch $S(0)$ via the following procedure: for each j, $b_j = 0$ indicates that the path extends from $S(j+1)$ to the first $S(j)$ switch to which it is directly connected, and $b_j = 1$ indicates that the path extends to the second $S(j)$ switch. Note that regardless of which input of $S(n)$ is selected, this path arrives at the same $S(0)$ switch, which is given the number i. See also the figure below the sample data for details of how the numbering works.

Sometimes multiple connections are needed at the same time. In order to avoid interference, each of the inputs and outputs of all switches $S(j)$ ($0 \leq j \leq n$) can be used by at most one connection. Given a set of connection requests, can you find connection paths for each request such that the connection paths are disjoint?

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 ($1 \leq n \leq 16$) and m ($1 \leq m \leq 2^n$): the layer of the outermost switch and the number of connection requests.

- m lines, each with two integers a_i and b_i ($0 \leq a_i, b_i < 2^n$). Each such line represents a connection request from input number a_i of $S(n)$ to output number b_i. You may assume that the integers a_i are pairwise distinct, and the integers b_i are pairwise distinct as well.

Output

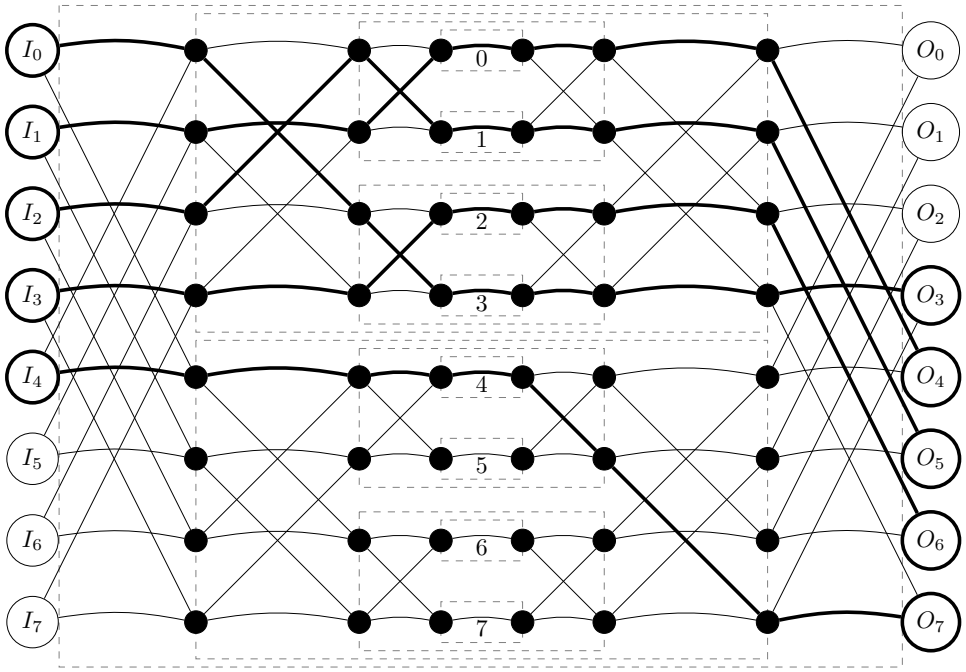
Per test case:

- One line with m integers $s_1, \ldots, s_m$, where s_i is the number of the $S(0)$ switch through which the connection of input a_i to output b_i is established.

The connection paths should be disjoint. You may print any valid solution, and you may assume that there is at least one valid solution.

Sample in- and output

Input	Output
2 1 1 0 1 3 5 0 3 1 4 2 5 3 6 4 7	0 3 0 1 2 4



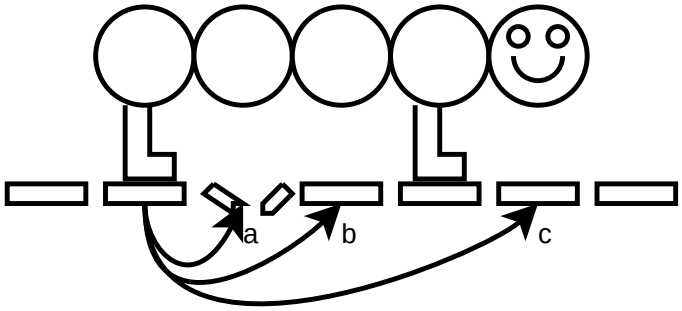
A possible connection scheme for the second sample case, with used cables in bold.

J Wormly

Jonly is writing his first computer game. For the opening scene he wants to have the main character, Wormly, cross Bridgely, the bridge. Wormly is a worm made of b equal circular bubbles and l legs. At all times each leg has to be under one of the bubbles, and under each bubble there can be at most one leg. Bridgely was supposed to be composed of n planks with the width of each plank equal to the diameter of each of Wormly's bubbles. However, some of the planks are missing.

At every moment, Wormly can do exactly one of the following:

- Move one of its legs forward over any number of (possibly missing) planks. After the move, the leg should be on a plank and underneath one of Wormly's bubbles. A leg isn't allowed to overtake other legs.
- Move all of its bubbles forward one plank while its legs remain on the same planks. After the move each leg must still be under one of Wormly's bubbles.



In this example, the only possible move for the last leg is to position b. (The plank at position a is missing, so the leg cannot move there. To get to position c, the last leg would have to overtake the first leg.) Also, in this example, moving all the bubbles forward is not allowed because Wormly's last leg would end up without a bubble over it.

Now Jonly is wondering how long the animation takes until Wormly reaches the end of Bridgely. Initially Wormly's bubbles are directly above the leftmost b planks of the bridge and its legs are on the leftmost l planks. At the end of the animation Wormly's bubbles have to be directly above the rightmost b planks and its legs have to be on the rightmost l planks.

The left- and rightmost l planks of Bridgely are not missing.

Input

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

- One line with three integers l, b and n ($1 \leq l \leq b \leq n \leq 1\,000\,000$): the number of legs, the number of bubbles and the length of the bridge respectively.
- One line with a string consisting of n characters, either '`0`' or '`1`', describing Bridgely. A one indicates a plank and a zero indicates a missing plank.

Output

Per test case:

- One line with an integer: the minimum number of steps it takes Wormly to cross Bridgely. If it is impossible to get across while satisfying the constraints, the line must contain "`IMPOSSIBLE`" instead.

Sample in- and output

Input	Output
3 1 2 2 11 2 3 5 11011 1 3 5 11011	1 IMPOSSIBLE 5