XXX Maraton Nacional de Programacion
Colombia - ACIS / REDIS 2016
ACM ICPC

Problems

(This set contains 11 problems; problem pages numbered from 1 to 23)

General Information

Unless otherwise stated, the following conditions hold for all problems.

Program name

1. Your source file (your solution!) must be called <codename>.c, <codename>.cpp, <codename>.java or <codename>.py, as indicated below the problem title.

Input

1. The input must be read from standard input.
2. The input contains several test cases. Each test case is described using a number of lines that depends on the problem.
3. When a line of data contains several values, they are separated by single spaces. No other spaces appear in the input. There are no empty lines.
4. Every line, including the last one, has the usual end-of-line mark.
5. The end of input is indicated by the end of the input stream. There is no extra data after the test cases in the input.

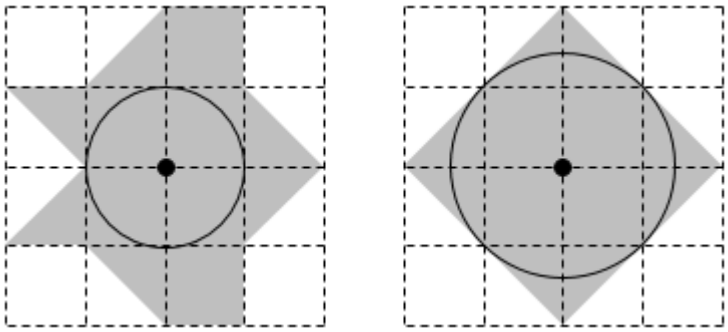
Output

1. The output must be written to standard output.
2. The result of each test case must appear in the output using a number of lines that depends on the problem.
3. When a line of results contains several values, they must be separated by single spaces. No other spaces should appear in the output. There should be no empty lines.
4. Every line, including the last one, must have the usual end-of-line mark.
5. After the output of all test cases, no extra data must be written to the output.
6. To output real numbers, round them to the closest rational with the required number of digits after the decimal point. Ties are resolved rounding to the nearest upper value.

A: ACIS, A Contagious vIruS

Source file name: `acis.c`, `acis.cpp`, `acis.java`, *or* `acis.py`
Author: A. Sotelo

Scientists from REDIS (*REsearch of DISeases*), a famous investigation center in Raccoon City, accidentally caused the mutation of a very contagious virus known as ACIS (*A Contagious vIruS*), just when they were manipulating ACIS' DNA. Raphael, the main researcher at REDIS, was infected with ACIS while he was treating inoculated rats. After that, all persons at REDIS were infected in less than an hour. Immediately he discovered the issue, Raphael contacted the Major, who decided to quarantine the largest possible circular region centered at REDIS that is totally inside Raccoon City, whose boundaries are described with a polygon.



The Major wants to know the maximum radius of such circular region. Can you help him?

Input

The input consists of several test cases. The first line of a test case contains a single integer N indicating the number of vertices of the polygon describing the boundaries of Raccoon City ($3 \leq N \leq 16$). The second line of a test case contains two blank-separated integers x_R and y_R ($0 \leq x_R \leq 50$, $0 \leq y_R \leq 50$) indicating the position (x_R, y_R) where REDIS is located. Then follow N lines: line i contains exactly two blank-separated integers x_i and y_i, where (x_i, y_i) is the position of the i-th vertex of the polygon describing the boundaries of Raccoon City ($0 \leq x_i \leq 50$, $0 \leq y_i \leq 50$). You may assume that there are not two vertices located at the same position, and that REDIS is located inside the polygon excluding its boundaries. The input ends with a line containing a single asterisk ('*').

The input must be read from standard input.

Output

For each test case, print a single line with a number indicating the radius of the largest possible circular region centered at REDIS that is totally inside Raccoon City. The answer should be formatted and approximated to three decimal places. The floating point delimiter must be '`.`' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.3712 is rounded to 78.371; 78.5766 is rounded to 78.577; 78.3745 is rounded to 78.375, etc.).

The output must be written to standard output.

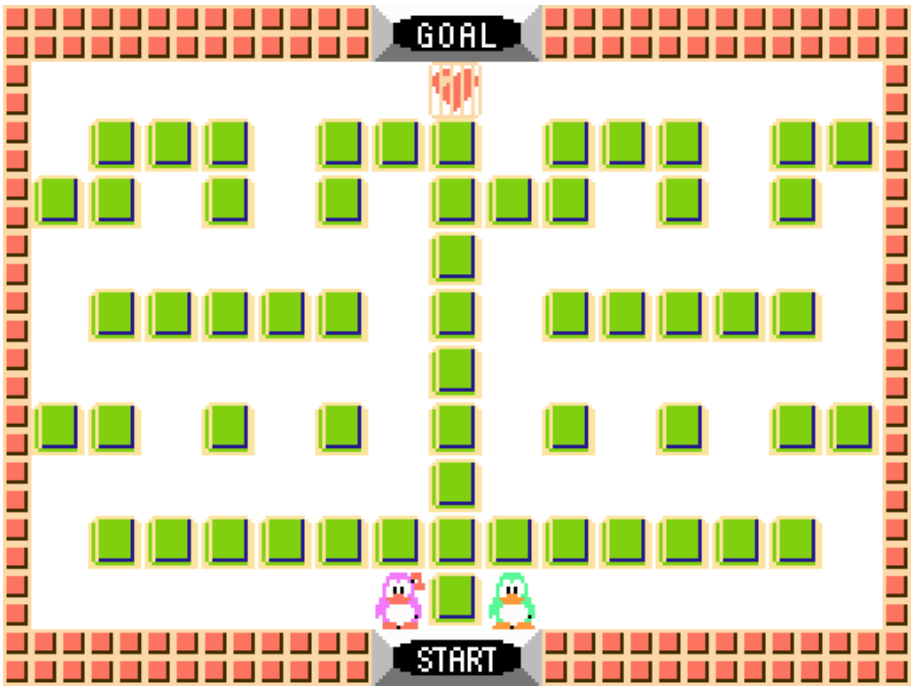
Sample Input	**Sample Output**
12 2 2 0 1 1 1 2 0 3 0 3 1 4 2 3 3 3 4 2 4 1 3 0 3 1 2 4 2 2 0 2 2 0 4 2 2 4 *	1.000 1.414

B: **Binary Land**

Source file name: `binary.c`, `binary.cpp`, *or* `binary.java`
Author: A. Sotelo

Gurin and Malon are a couple of penguins living in Binary Land, a marvelous country. They are trapped in a mystical maze, described as a grid with cells that are either free spaces or walls. Exactly one of the free spaces is designated as the *love cell*, having a nice heart inside a cage. Gurin and Malon are initially located at two free spaces inside the maze.

The maze is surrounded by walls, so no penguin can move outside it because, as everyone knows, penguins cannot move through walls.



Gurin (right) and Malon (left) trapped in the maze. North is upside and West is leftside.
Original image taken from Binary Land, Hudson Soft Co., Ltd.

Both penguins can move freely through the free spaces, until they meet at the love cell, where they can fall in love together. At any given time, a penguin can move from its current cell to an adjacent cell in one of four possible directions: north, south, east and west.

However, Gurin and Malon were cursed by an evil witch! If a penguin goes north or south, then the other must automatically go in the same direction; and, if a penguin goes east or west, then the other must automatically go in the opposite direction. As it was mentioned before, no penguin can move through a wall and additionally, both penguins can be in the same cell at any given time.

In detail, the curse works as follows:

- If a penguin has a free cell to the *north* and it moves one step to the *north*, then the other penguin must move one step to the *north* (at the same time). However, if the other penguin had a wall to the *north*, it must stay in its current cell.
- If a penguin has a free cell to the *south* and it moves one step to the *south*, then the other penguin must move one step to the *south* (at the same time). However, if the other penguin had a wall to the *south*, it

must stay in its current cell.

- If a penguin has a free cell to the *west* and it moves one step to the *west*, then the other penguin must move one step to the *east* (at the same time). However, if the other penguin had a wall to the *east*, it must stay in its current cell.
- If a penguin has a free cell to the *east* and it moves one step to the *east*, then the other penguin must move one step to the *west* (at the same time). However, if the other penguin had a wall to the *west*, it must stay in its current cell.

Each cursed move of both penguins takes exactly one unit of time. Given a maze, the coordinates of the love cell, and the initial coordinates of Gurin and Malon, what is the minimum amount of time in which both penguins can fall in love together?

Input

The input consists of several test cases. The first line of a test case contains two blank-separated integers R and C ($1 \leq R \leq 40$, $1 \leq C \leq 40$) indicating, respectively, the number of rows and columns of the maze (without the surrounding walls). The second line contains six blank-separated integers r_L, c_L, r_G, c_G, r_M, and c_M ($1 \leq r_L, r_G, r_M \leq R$, and $1 \leq c_L, c_G, c_M \leq C$) indicating the coordinates (r_L, c_L) of the love cell, the initial coordinates (r_G, c_G) of Gurin, and the initial coordinates (r_M, c_M) of Malon. Each of the next R lines contains C characters '`.`' or '`#`', where '`.`' represents a free space and '`#`' represents a wall. You may assume that the coordinates (r_L, c_L), (r_G, c_G) and (r_M, c_M) correspond to free spaces, and that the given maze is surrounded by walls.

The input must be read from standard input.

Output

For each test case, output a single line with the minimum amount of time in which both penguins can meet at the love cell or with the text '`NO LOVE`' if it is impossible for them to meet at the love cell.

The output must be written to standard output.

Sample Input	**Sample Output**
10 15 1 8 10 9 10 7 ....###.###.###.## ##.#.#.###.#.#. ....#....... .#####.#.#####. ....#....... ##.#.#.#.#.#.## ....#....... .############. ....#....... 3 3 1 2 3 2 3 2 ... .#. ... 3 3 1 2 3 2 3 2 ... ### ... 3 3 3 2 3 2 3 2 ... ### ...	31 4 NO LOVE 0

C: Castaways

Source file name: `castaways.c`, `castaways.cpp`, `castaways.java`, *or* `castaways.py`
Author: R. Cardoso

Alexa and Bob are castaways survived to a shipwreck arriving to a lonely island. This island is relatively near to the mainland but there is no easy way to send an alarm message to tell what happened.

Alexa and Bob arrived to the island coast clinging to a wooden cargo box that -besides being a lifesaver- contained a lot of wooden sticks of different sizes and a fabric roll. Since they are experimented windsurfers, they are planning to build two windsurf boards that they could use to get off the island sailing to mainland. Each windsurf board may be made with a side plank of the wooden box. And the board must be propelled by the wind, so that they want to build sails for the two boards. The sails may be constructed with some wooden sticks forming a frame and a piece of fabric from the roll.

But in order to build a wind-surf sail, Alexa prefers a triangular one and Bob a rectangular one. There are arguments to support that triangles are better than rectangles when one talks about windsurf sails, and now it is not possible that Alexa and Bob come to a common solution. They know, however, that it is better to build sails with a big area, so that the wind may propel the board more effectively.

To avoid further discussions they come to the next problem to solve: design the largest pair of triangular and rectangular sails, given the sizes of the wooden sticks (there is enough fabric in the roll to make any sail). When they say ‘‘the largest pair’’ they mean a triangle and a rectangle whose areas, together, are as big as possible. Note that it can be that such a maximal pair could be formed by a triangle and a void rectangle, by a void triangle and a rectangle, or by a void triangle and a void rectangle, because in this way the total area may be maximized.

Lets illustrate the situation with some examples. Suppose first that the box contains only 3 sticks, of lengths 3, 4 and 5 cm. Then the castaways can build only a triangular sail, with area 6 cm^2. And if the box contains 7 sticks, of lengths 1, 4, 2, 1, 3, 3, 2 cm, they may build a pair consisting of a rectangular sail of 6 cm^2 and a void triangle. Finally, if there are 9 sticks in the box, each one of length 10 cm, they can build an equilateral triangle and a square as their largest pair, with total area 143 cm^2 (neglecting fractions of cm^2).

You must write a program to help the castaways design their sails.

Input

The input consists of several test cases. The first line of each test case contains a single integer N indicating the number of wooden sticks in the box ($3 \leq N \leq 256$). Then there are N lines, each one with one single integer x ($1 \leq x \leq 256$), representing the sizes of the N wooden sticks, measured in centimeters.

The input must be read from standard input.

Output

For each test case, print a single line with an integer indicating the area of the largest pair (as defined above), in squared centimeters (neglecting decimals), that may be constructed with the N given sticks.

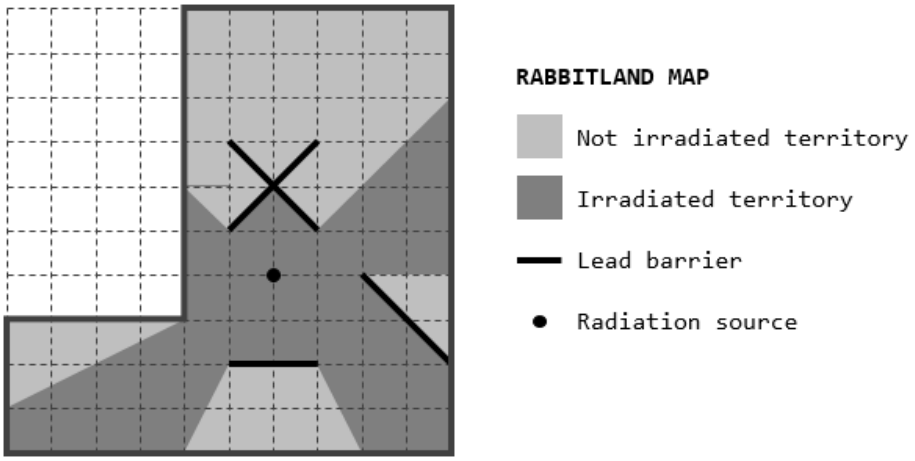
The output must be written to standard output.

Sample Input	**Sample Output**
3 3 4 5 7 1 4 1 2 3 2 3 9 10 10 10 10 10 10 10 10 10 3 1 1 8	6 6 143 0

D: Radiation Alert in Rabbitland

Source file name: `radiation.c, radiation.cpp`, *or* `radiation.java`
Author: R. García, A. Sotelo

A radiation source was recently discovered by some scientists of Rabbitland while they were digging a burrow. Immediately, King Rabbit ordered the reinforcement of Rabbitland's boundary using lead barriers, and the installation of some additional lead barriers inside Rabbitland. Rabbitland's boundary is defined with a polygon, and each lead barrier is described with a line segment.



Each lead barrier blocks the radiation, protecting some regions inside Rabbitland. You have been hired by King Rabbit to calculate the percentage of Rabbitland's territory that is being irradiated by the radiation source.

Input

The input consists of several test cases. The first line of a test case contains two blank-separated integers N and M, where N is the number of vertices of the polygon describing the boundary of Rabbitland ($3 \leq N \leq 16$), and M is the number of additional barriers installed inside Rabbitland ($0 \leq M \leq 20$). The second line of a test case contains two blank-separated integers x_S and y_S ($0 \leq x_S, y_S \leq 25$) indicating the position (x_S, y_S) where the radiation source was discovered. Then follow N lines: line i contains exactly two blank-separated integers x_i and y_i, where (x_i, y_i) is the position of the i-th vertex of the polygon describing the boundary of Rabbitland ($0 \leq x_i, y_i \leq 25$). Finally follow M lines: line j contains exactly four blank-separated integers x_{1j}, y_{1j}, x_{2j} and y_{2j}, where (x_{1j}, y_{1j}) and (x_{2j}, y_{2j}) are the positions of the endpoints of the line segment that describes the j-th additional lead barrier installed inside Rabbitland ($0 \leq x_{1j}, y_{1j}, x_{2j}, y_{2j} \leq 25$, $(x_{1j}, y_{1j}) \neq (x_{2j}, y_{2j})$). You may assume that the radiation source and every lead barrier are completely inside the polygon including its boundary.

The input must be read from standard input.

Output

For each test case, print a single line with a number indicating the percentage of Rabbitland's territory that is being irradiated by the radiation source, followed by the percentage sign ('%'). The answer should be formatted and approximated to three decimal places. The floating point delimiter must be '.' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.3712 is rounded to 78.371; 78.5766 is rounded to 78.577; 78.3745 is rounded to 78.375, etc.).

The output must be written to standard output.

Sample Input	**Sample Output**
6 4 6 6 4 0 10 0 10 10 0 10 0 7 4 7 5 8 7 8 5 5 7 3 5 3 7 5 8 6 10 8 4 3 2 2 0 0 0 4 4 4 4 0 0 1 4 1 1 1 3 1 1 2 3 2 4 1 0 0 0 0 4 0 4 4 0 4 0 0 4 4	50.000% 75.000% 100.000%

E: Eclipsing Gianik Star

Source file name: `gianik.c, gianik.cpp`, *or* `gianik.java`
Author: A. Sotelo

Gianik is a giant pink star in Canis Major constellation at an approximate distance of 700 light years from our Solar System. Each planet of Gianik's planetary system follows a trajectory described by a circular orbit centered at Gianik, whose coordinates $(x(t), y(t))$ at time t obeys the parametric equations

$$x(t) = \rho \cdot \cos(\alpha + \beta \cdot t)$$
$$y(t) = \rho \cdot \sin(\alpha + \beta \cdot t)$$

where ρ is a positive integer denoting the radius of the circular orbit, and $\alpha + \beta \cdot t$ is a linear function with integer coefficients α,β describing the angle subtended by the planet's trajectory from time 0 to time t. All angles are measured in degrees (°), where one full rotation around Gianik takes 360°.

An eclipse occurs when Gianik and two distinct planets are located at collinear coordinates at the same time t, so that Gianik is not visible from the farthest planet because it is eclipsed by the other planet. May you determine the minimum non-negative integer t such that an eclipse occurs at time t?

Input

The input consists of several test cases. The first line of each test case contains a single integer N indicating the number of planets of Gianik's planetary system ($2 \leq N \leq 300$). Each of the next N lines contains three blank-separated integers ρ, α and β, indicating the parameters that describe the planet's trajectory around Gianik according to the statement ($1 \leq \rho \leq 1000$, $-1000 < \alpha < 1000$, $-1000 < \beta < 1000$). You may assume that the orbits of all N planets have distinct radiuses.

The input must be read from standard input.

Output

For each test case, print a single line with a non-negative integer indicating the minimum time in which an eclipse occurs in Gianik's planetary system. If no eclipse occurs at any non-negative time, then print the text '`GIANIK IS NEVER ECLIPSED`'.

The output must be written to standard output.

Sample Input	**Sample Output**
2 10 90 2 20 0 4 2 10 90 4 20 0 2 2 10 90 2 20 0 2	45 135 GIANIK IS NEVER ECLIPSED

F: Funny Cardiologist

Source file name: `funny.c`, `funny.cpp`, *or* `funny.java`
Author: A. Sotelo

Dr. Zoidberg is a famous cardiologist and comedian that likes to make unpleasant jokes. One of his favorite jokes is to adulterate cardiograms to make people believe they will die. A *cardiogram* is defined as the diagram of a polyline described with a list of N points $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ with ascending x-coordinates, whose line segments are drawn between consecutive points. The *length* of a cardiogram is defined as the sum of the lengths of each segment drawn on it:

$$\sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

The technique used by Dr. Zoidberg to adulterate a cardiogram consists of removing exactly K points from its polyline, where K depends on the seriousness of the patient. Of course, there could be several ways to do that, but the joke will be the best (in Zoidberg's opinion!) if the resulting plot is as short as possible in the sense that it becomes a good approximation to a straight line, that is, if it makes the patient believe he or she is close to death. To avoid suspicion, Dr. Zoidberg does not remove neither the first point nor the last point from the polyline.

The cardiogram of a healthy patient.
Length: 36.393.

The first cardiogram removing some $K = 2$ points.
Length: 24.285.

The first cardiogram removing some $K = 9$ points.
Length: 20.000.

The last cardiogram corresponds to a patient who may believe he or she possibly will die. Dr. Zoidberg wants to know the minimum length that can be attained adulterating a cardiogram described by the polyline with points

$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, removing exactly K points from that polyline. May you help him?

Input

The input consists of several test cases. The first line of a test case contains two blank-separated integers N and K, where N is the number of points on the polyline describing the cardiogram ($2 \leq N \leq 256$), and K is the number of points that Dr. Zoidberg must remove from that polyline ($0 \leq K \leq N-2$). Then follow N lines: line i contains exactly two blank-separated integers x_i and y_i, where (x_i, y_i) is the position of the i-th point of the polyline describing the cardiogram ($0 \leq x_i < 1000$, $-1000 < y_i < 1000$). You may assume that the points of the polyline have ascending x-coordinates (i.e., $x_1 < x_2 < \cdots < x_n$).

The input must be read from standard input.

Output

For each test case, print a single line with the minimum length that can be attained adulterating the cardiogram using Dr. Zoidberg's technique. The answer should be formatted and approximated to three decimal places. The floating point delimiter must be '`.`' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.3712 is rounded to 78.371; 78.5766 is rounded to 78.577; 78.3745 is rounded to 78.375, etc.).

The output must be written to standard output.

Sample Input	**Sample Output**
11 2 0 0 5 0 6 -2 7 3 8 -3 9 0 11 0 12 1 13 -2 14 0 20 0 11 9 0 0 5 0 6 -2 7 3 8 -3 9 0 11 0 12 1 13 -2 14 0 20 0	24.285 20.000

G: Christmas Lights

Source file name: `lights.c, lights.cpp, or lights.java`
Author: R. García

Bill has a Christmas Lights String to decorate his house, made with K lights $L[1], L[2], \ldots, L[K]$ attached in sequence to a wire. The behavior of each light is determined by a programmable controller connected to the wire, turning *on* and *off* lights at every second.

Bill has programmed the controller to change the state of the lights during M seconds. He defines a pair of numbers a_i, b_i with $a_i \leq b_i$, for each second i ($1 \leq i \leq M$). At second 0, the string of lights is initialized with a random *initial configuration* (some lights *on* and other lights *off*). At each second i, from 1 to M, the state of all lights in $L[a_i \,..\, b_i]$ is simultaneously switched from *on* to *off* and vice versa. However, Bill added a curious little feature to the controller's algorithm: whenever the ends $L[a_i]$ or $L[b_i]$ are *off*, just before the above-described switching takes place at time i, some more lights in the string can switch states at moment i. In particular, if $L[a_i]$ is *off* and there is a light, say at l_i, to the left of a_i that is *on* (and all the lights between l_i and a_i are *off*), then the lights in the interval $L[l_i \,..\, a_i - 1]$ will also switch states at moment i. Similarly, if $L[b_i]$ is *off* and there is a light, say at r_i, to the right of b_i that is *on* (and all the lights between b_i and r_i are *off*), then the lights in the interval $L[b_i + 1 \,..\, r_i]$ will also switch states at moment i.

Suppose that a light turned *on* is represented with '`1`' and a light turned *off* is represented with '`0`'. For example, consider $K = 18$, $M = 5$, $a_1 = 5$, $b_1 = 12$, $a_2 = 10$, $b_2 = 11$, $a_3 = 5$, $b_3 = 8$, $a_4 = 3$, $b_4 = 6$, $a_5 = 1$, and $b_5 = 17$, with initial configuration `000110010011100000`. Note that the state of all lights at each second is:

- `000110010011100000` at second 0.
- `000101101100100000` at second 1.
- `000101101011000000` at second 2.
- `000010010011000000` at second 3.
- `001101100011000000` at second 4.
- `110010011100111110` at second 5.

After several days of operation, Bill suspects that he has created a truly awesome algorithm. For this purpose, he would like to run multiple trials, with different initial configurations and parameters a, b, but he is afraid the lights will break due to heavy abuse. Can you help him in building an algorithm for finding the final state of all lights at second M after each trial?

Input

The first line of the input contains a positive integer T indicating the number of test cases. The first line of a test case contains two blank-separated integers K and M ($2 \leq K \leq 10^6$, $0 \leq M \leq 10^4$) indicating, respectively, the number of lights in the string and the number of seconds to consider. The second line contains a hexadecimal string (using digits '`0123456789ABCDEF`') without leading zeros, describing the initial configuration of lights if it is written in binary notation. If the given hexadecimal string requires less than K bits in binary notation, then complete it with leading zeros to reach K digits. Finally follow M lines: line i contains exactly two blank-separated integers a_i and b_i describing the parameters controlling the behavior of the lights at second i ($1 \leq i \leq M$, $1 \leq a_i \leq b_i \leq K$).

The input must be read from standard input.

Output

For each test case, print a single line with a hexadecimal string (using digits '`0123456789ABCDEF`') without leading zeros, describing the state of all lights at second M. You must use the same notation used to codify the initial configuration of lights.

The output must be written to standard output.

Sample Input	**Sample Output**
3 18 5 64E0 5 12 10 11 5 8 3 6 1 17 18 0 0 18 1 0 13 16	3273E 0 3C

H: socialhare

Source file name: `hare.c`, `hare.cpp`, *or* `hare.java`
Author: C. Rocha

Little Elisa is learning her first words. With the help of her parents and friends, she has been introduced to word search puzzles in order to practice and acquire new vocabulary. A search word puzzle consists of a rectangular arrangement of lowercase English characters and a list of words (each consisting of lowercase English characters too), with the goal of finding as many words from the list as possible in the arrangement of letters.

However, little Elisa thinks of word search puzzles in a more relaxed way: she considers a word w to occur in a rectangular arrangement of letters if she can find any permutation of w in the rectangular arrangement (horizontal, vertical or diagonally).

Since little Elisa is a bit of an anarchist and ''women always get the last word in every argument'', her parents and friends have designed some word search puzzles so that Elisa can have fun her own way. They are asking for help in checking if Elisa's solutions are correct. Can you help?

Input

The input consists of several test cases. The first line of a test case contains three-blank separated integers R, C, W ($1 \leq R, C, W \leq 100$) denoting the number of rows R and columns C of the rectangular arrangement of characters, and the number of words W to search for in the game. Each of the following R lines consists of exactly C lowercase English characters. Then, the next W lines describe the list of words to search for in the arrangement of characters. Each of these lines contains a non-empty sequence of at most 100 lowercase English characters.

The input must be read from standard input.

Output

For each test case, output the maximum number of words from the given list of W words that appear, according to Elisa's rules, in the rectangular arrangement of $R \times C$-characters.

The output must be written to standard output.

Sample Input	**Sample Output**
10 11 3 entanglerci alkaksckasj qwicasasqwb asososqweqn gasxahqaqsa psposrtgppa uuusasosaah zxcvbnmclwq xscdvfbghqr fsdddllslaq socialhare rectangle bear 2 2 6 ab cd ba ac aa da cb ae	1 4

I: Water troubles

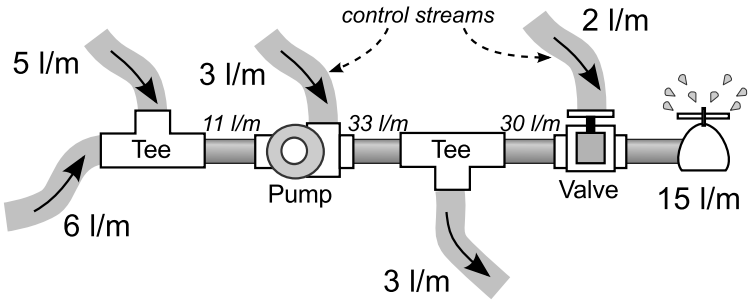
Source file name: `water.c, water.cpp, water.java, or water.py`
Author: M. Sánchez

Because of climate change, the little town of Minca in the Sierra Nevada is facing serious problems with its water supplies. They were used to get water from the abundant streams, brooks, and small rivers coming down the mountain, but now all of these carry a lot less water. On top of that, their coffee fields depend on irrigation machines and require precise amounts of water in order to survive.

Some ingenious hydraulic engineers that heard about Minca's problems decided to pay them a visit to help them save their delicate crops. They brought with them a series of gadgets and pipe fittings to divert water streams and gather just the right amount of water that coffee requires. None of those devices depend on electricity or fuel, making them perfect for usage in the mountain. The first kind of device was just a tee pipe: depending on the way it is installed, it combines two streams or divides a stream in two parts (which may transport different amounts). The second kind of device was a water controlled water pump: it takes a controlling stream of water on one of its input fittings and augments the other input stream by a factor that depends on the controlling stream. Finally, there were hydraulic valves that reduce input streams by a factor that also depends on some controlling water stream.

Everything was joy in Minca because these ingenious devices were going to help the town properly operate its irrigation machines. Unfortunately, the engineers quickly discovered that, in order to send the proper amount of water to each field, they would have to combine in creative ways their machines and the flexible and durable hoses that they use to take water from the streams and to dispose water on them when unused.

In the following figure you can see how they solved the problem of getting 15 liters of water per minute to some field, considering that they had with them hoses for 6, 3, 2, 5, 20 and 3 l/m. The first tee pipe receives 5 l/m and 6 l/m from the corresponding hoses. Next, a hydraulic pump controlled by a 3 l/m stream feeds 33 l/m to the next tee pipe, which discards 3 l/m. Finally, the valve is controlled by a 2 l/m stream, taking down its output to the 15 l/m that the field and its irrigation system requires. The 20 l/m hose was not used. It were used the hoses for 5, 6, 3, 3 and 2 l/m, in that order.



Your task is to help Mincans and the hydraulic engineers to assemble their devices and hoses in linear sequences to save the fields. For this, you will receive the required amount of water required by a given field, and the capacity of the hoses available to grab and dispose water from the brooks and small rivers.

For simplicity, assume that engineers have an unlimited amount of every type of device, that there is a single irrigation machine in the coffee field, that a tee device that divides a stream always sends the extra water back to a brook, and that the water flows must always be integers. Also, you do not have to consider the pipes and tubes used to connect the devices between them, or to connect the devices to the irrigation machine.

Input

The input consists of several test cases. Each test case is described by a single line containing $H + 1$ blank-separated integers $T, c_1, c_2, \ldots, c_H$, where T indicates the amount of water required by a certain field in the town ($1 \leq T \leq 10^{15}$), H is the number of hoses ($1 \leq H \leq 7$), and $c_1, c_2, \ldots, c_H$ indicate the capacity of the hoses available to direct the water ($1 \leq c_i \leq 50$ for each $1 \leq i \leq H$).

The last test case is followed by a line containing a single zero value.

The input must be read from standard input.

Output

For each test case, print a line with a single integer. This integer should indicate the amount of water that may be sent into the field with an optimal arrangement of the hoses and the devices. If it is impossible to find an arrangement that sends the exact quantity of water required, your program should find the closest quantity that is superior to the requirement, or 0 if it is impossible to reach the required amount.

The output must be written to standard output.

Sample Input	**Sample Output**
15 6 3 2 5 20 3 15 20 5 6 3 3 2 15 5 6 3 3 2 10 1 2 3 8 6 4 5 14 4 3 7 5 11 3 3 3 6 3 3 0	15 15 15 0 9 14 12 6

J: Wildcards

Source file name: `wildcards.c`, `wildcards.cpp`, *or* `wildcards.java`
Author: A. Echavarría

Alice and Bob are playing a game: Alice selects a text t and a word w, and then Bob has to say how many times w occurs in t. However, after a while, Alice realizes that this version of the game is too boring for Bob and decides to make a modification: in her new version of the game, the wildcard symbol '`?`' can occur in w any number of times. Each occurrence of '`?`' in w can be matched with any character in t.

In the new version of the game, for instance, if the text is t = `banana` and the word is w = `?a?`, then w occurs twice in t: at position 0 matching `ban` and at position 2 matching `nan`. Notice that matches can overlap.

Can you write a program to help Bob solve this new game?

Input

The input consists of several test cases, each one defined by two lines. The first line contains the text t and the second line contains the word w. The text t consists of lowercase letters from the English alphabet ($1 \leq |t| \leq 10^5$), and the word w consists of lowercase letters from the English alphabet and the wildcard character '`?`' ($1 \leq |w| \leq 10^5$). It is guaranteed that there will be at most k wildcard characters in w, where $0 \leq k \leq \min\left(|w|, 10^6/|t|\right)$.

The input must be read from standard input.

Output

For each test case, print a line with one integer denoting the number of times w appears in t if each wildcard character matches any character in t.

The output must be written to standard output.

Sample Input	Sample Output
banana ?a? bananas ?a? abide a??d abide a?d abracadabra a?a acisredis ?b acisredis ?? icpc world?finals	2 3 1 0 2 0 8 0

K: **Bank Robbery**

Source file name: `bank.c, bank.cpp`, *or* `bank.java`
Author: C. Rocha

Arsène Lupin is a gentleman thief and a master of disguise; he has been responsible for heists no right-minded individual would believe possible. He is also, very much, the ladies' man.

Lupin is about to drop everything he is currently doing to come to the aid of some of his friends who are planning a bank robbery in the infamous Kingdom of Aksum: his friends have identified the location of banks they are willing to rob, as well as the location of the police stations that serve the city. As a matter of fact, they have come up with a map of the entire city in which bidirectional roads connecting sites and traveling times between sites have been detailed.

Despite the criminal nature of his activities, Lupin has a strict code he follows in order to avoid tainting his reputation: he has never been caught by the authorities. In order to help his friends and, at the same time, keep his well-earned reputation, Lupin is wondering which banks can be robbed so that they are the ones furthest away from any police station serving the Kingdom of Aksum.

Input

The input consists of several test cases. Each test case begins with 4 blank-separated integer numbers N, M, B, P ($1 \leq N \leq 1\,000$, $0 \leq M$, $1 \leq B \leq N$, $0 \leq P < N$) denoting, respectively, the number of sites in the city, the number of roads in the city, the number of banks in the city, and the number of police stations in the city. The next M lines contain each three blank-separated integers U, V, T ($0 \leq U < N$, $0 \leq V < N$, $U \neq V$, $0 \leq T \leq 10\,000$) denoting that there is a road between sites U and V which takes T time units to transit. The next line contains B blank-separated and pairwise distinct site numbers identifying the location of banks. If $P \neq 0$, then follows a line with P blank-separated and pairwise distinct site numbers identifying the location of police stations. You can assume that a bank and a police station are never located at the same site.

The input must be read from standard input.

Output

For each test case, output two lines. The first line should contain two blank-separated figures S, E denoting, respectively, the number of banks furthest away from any police station and the minimum time it would take to transit from any police station to these banks. If E is not an integer number, then output '*' instead. The second line should contain S blank-separated integers, in ascending order, corresponding to the sites where banks are located with minimum time from any police station being exactly E time units.

The output must be written to standard output.

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