acm International Collegiate Programming Contest

2009 South Western European Regional Contest Problems

Sponsored by

IBM event sponsor

CORITEL An Accenture Company FUJITSU tuenti

Hosted by

Facultad de Informática

Facultad de Informática. Universidad Complutense de Madrid

A
Trick or Treat

Johnny and his friends have decided to spend Halloween night doing the usual candy collection from the households of their village. As the village is too big for a single group to collect the candy from all houses sequentially, Johnny and his friends have decided to split up so that each of them goes to a different house, collects the candy (or wreaks havoc if the residents don't give out candy), and returns to a meeting point arranged in advance.

There are n houses in the village, the positions of which can be identified with their Cartesian coordinates on the Euclidean plane. Johnny's gang is also made up of n people (including Johnny himself). They have decided to distribute the candy after everybody comes back with their booty. The houses might be far away, but Johnny's interest is in eating the candy as soon as possible.

Keeping in mind that, because of their response to the hospitality of some villagers, some children might be wanted by the local authorities, they have agreed to fix the meeting point by the river running through the village, which is the line $y = 0$. Note that there may be houses on both sides of the river, and some of the houses may be houseboats ($y = 0$). The walking speed of every child is 1 meter per second, and they can move along any direction on the plane.

At exactly midnight, each child will knock on the door of the house he has chosen, collect the candy instantaneously, and walk back along the shortest route to the meeting point. Tell Johnny at what time he will be able to start eating the candy.

Input

Each test case starts with a line indicating the number n of houses ($1 \leq n \leq 50\,000$). The next n lines describe the positions of the houses; each of these lines contains two floating point numbers x and y ($-200\,000 \leq x, y \leq 200\,000$), the coordinates of a house in meters. All houses are at different positions.

A blank line follows each case. A line with $n = 0$ indicates the end of the input; do not write any output for this case.

Output

For each test case, print two numbers in a line separated by a space: the coordinate x of the meeting point on the line $y = 0$ that minimizes the time the last child arrives, and this time itself (measured in seconds after midnight). Your answer should be accurate to within an absolute or relative error of 10^{-5}.

Sample Input

```
2
1.5 1.5
3 0

1
0 0

4
1 4
4 4
-3 3
2 4

5
4 7
-4 0
7 -6
-2 4
8 -5

0
```

Sample Output

```
1.500000000 1.500000000
0.000000000 0.000000000
1.000000000 5.000000000
3.136363636 7.136363636
```

B
Working at the Restaurant

Last night, Tom went on a date with a really nice girl. However, he forgot to take his credit card with him and he had no cash in his wallet, so he ended up working at the restaurant to pay for the bill. His task is to take plates from the waiter when he comes from the tables, and pass them along when the diswasher requests them. It is very important for the plates to be washed in the same order as they are brought from the tables, as otherwise it could take too long before a plate is washed, and leftover food might get stuck. Trying to hold all the plates in his hands is probably not a great idea, so Tom puts them on a table as soon as the waiter hands them over to him, and picks them up from the table again when the time comes to pass them along to the dishwasher. There is space for only two piles of plates on the table, which will be referred to as pile 1 and pile 2. There is only one table Tom can use.

Tom won last year's SWERC, so he is certainly capable of optimizing for efficiency. You have to output a transcript of one possible way in which Tom might decide to organize the plates on the table during the process, given the sequence of plates and requests he receives.

Input

The input has several test cases. Each case begins with a line containing a number N $(1 \leq N \leq 1\,000)$, followed by N lines, which contain either `DROP m` or `TAKE m`, where $m > 0$ is the number of plates to take or drop. `DROP m` represents that the next event is the waiter bringing m plates to Tom, so he has to drop them on the table, while `TAKE m` represents that the next event is Tom taking m plates from the table and passing them along in the right order. You can assume that he never receives a `TAKE m` instruction when there are fewer than m plates on the table, and that the sum M of all values of m corresponding to `DROP` operations does not exceed 100 000. Note that there might be plates left on Tom's table when the last request is issued, as Tom might be relieved of his duty to stay until the restaurant closes.

The input ends with a line with $N = 0$, which must not be processed.

Output

For every test case, the output will be a series of lines describing the operations to be performed with the plates. The content of each line will be one of the following:

- `DROP 1 m` (`DROP 2 m`), $m > 0$, if Tom needs to take a plate from the waiter, drop it on top of pile 1 (pile 2), and repeat this operation m times in total.
- `TAKE 1 m` (`TAKE 2 m`), $m > 0$, if Tom needs to take a plate from the top of pile 1 (pile 2), pass it along to the dishwasher, and repeat m times in total.
- `MOVE 1->2 m` (`MOVE 2->1 m`), $m > 0$, if Tom needs to take a plate from the top of pile 1 (pile 2), drop it on top of pile 2 (pile 1), and repeat m times in total.

You must output at most $6N$ lines, and the total number of movements of plates in your transcript (that is, the sum of the `m`'s printed in your output, for all three kinds of operations), must be at most $6M$, as otherwise Tom won't be able to cope with all the work.

Note that Tom must obey the commands in the same order as they are issued. This means that, if he receives a TAKE `m` command, he must perform a certain number of MOVE and TAKE operations such that the sum of the numbers of plates **taken** adds up exactly to `m` before performing the operations corresponding to the next command; and if he receives a `DROP m` command, he must perform a number of DROP or MOVE operations for which the sum of the numbers of plates **dropped** adds up exactly to `m` before performing the operations corresponding to the next command.

Of course, it is also forbidden to take plates from the waiter or pass them along to the dishwasher in the absence of the corresponding order.

There **must** be an empty line between the outputs of different cases.

Any solution satisfying these conditions will be accepted.

Sample Input

```
3
DROP 100
TAKE 50
TAKE 20
3
DROP 3
DROP 5
TAKE 8
0
```

Sample Output

```
DROP 2 100
MOVE 2->1 100
TAKE 1 50
TAKE 1 20

DROP 2 3
DROP 2 5
MOVE 2->1 8
TAKE 1 8
```

C
Lights

John has n light bulbs and a switchboard with n switches; each bulb can be either on or off, and pressing the $i-$th switch changes the state of bulb i from on to off, and viceversa. He is using them to play a game he has made up. In each move, John selects a (possibly empty) set of switches and presses them, thus inverting the states of the corresponding bulbs. After exactly m moves, John would like to have the first v bulbs on and the rest off; otherwise he loses the game. There is only one restriction: he is not allowed to press the same *set* of switches in two different moves.

This is quite an easy game, as there are lots of ways of winning. This has encouraged him to keep playing different winning games, and now he is intent on trying them all. Help him count how many ways of winning there are. Two games are considered the same if, after a reordering of the moves in one of them, at every step the same set of switches is pressed in both of them.

For example, if $n = 4$, $m = 3$, and $v = 2$, one possible winning game is obtained by pressing switches 1, 2 and 4 in the first move, 1 and 3 in the second one, and 1, 3 and 4 in the last one. This is considered equivalent to, say, first pressing 1 and 3; then 1, 2, 4; and then 1, 3, 4.

Input

The input has at most 500 lines, one for each test case. Each line contains three integers n $(1 \leq n \leq 1\,000)$, m $(1 \leq m \leq 1\,000)$, and v $(0 \leq v \leq n)$. The last line of input will hold the values `0 0 0` and must not be processed.

Output

Print one line for each test case containing the number of ways John can play the game, modulo the prime 10 567 201.

Sample Input

```
3 3 1
6 4 0
6 4 3
0 0 0
```

Sample Output

```
7
10416
9920
```

D
Darts

After a long week of work at the ICPC Headquarters, Bill and his friends usually go to a small pub on Friday evenings to have a couple of beers and play darts. All of them are well aware of the fact that their ability at darts decreases at the same rate as the amount of beer left in their mugs.

They always play 501, one of the easiest games. Players start with a score of N points (typically, $N = 501$, hence the name) and take turns throwing darts. The score of each player decreases by the value of the section hit by the dart, unless the score becomes negative, in which case it remains unchanged. The first player to reach a score of 0 wins. The figure below shows the dartboard with which the game is played.

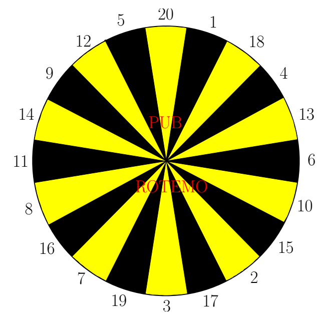


Figure 1: Dartboard

As the clock ticks closer to midnight and they start running out of beer, everyone wonders the same: is it worth trying to aim the dart at a specific section? Or is it better just to throw the dart at a random section on the dartboard? You are asked to deal with the question by finding out what would happen if two players (A and B) applying these two different strategies were to play against each other:

- Player A throws the darts at random, and consequently they land with equal probability in each of the sections of the dartboard.
- If Player B aims at a certain section, the dart has the same probability of landing in the correct one as in each of the two adjacent ones (the neighbouring regions to the left and right). Moreover, he is completely aware of his ability and sober enough to aim at the section that maximizes his probability of winning.

Given the initial score of both players, can you determine the probability that the first player wins? Of course, being the first to throw a dart might be advantageous, so the answer depends on who plays first.

Input

The input consists of a series of lines, each containing an integer N ($1 \le N \le 501$), the initial score of both players. A case with $N = 0$ marks the end of the input and must not be processed.

Output

For each number in the input, your program should output a line containing two real numbers: the probability that A wins if A throws the first dart, and the probability that B wins if B throws the first dart. Your answers should be accurate to within an absolute or relative error of 10^{-8}.

Sample Input

```
5
100
0
```

Sample Output

```
0.136363636364 0.909090909091
0.072504908290 0.950215081962
```

E
Genetics

A colony of alien bacteria has recently been discovered close to a crater in New Mexico. Dr. Poucher is in charge of the scientific team at the ICPC BioLab committed to the study of the alien DNA structure. We briefly sketch their discoveries here.

Alien DNA molecules have the structure of a circular sequence. Each sequence is composed of nucleotides. There are 26 different types of nucleotides, and each of them can occur in two faces. It is very important to remark that in any given alien DNA molecule, every nucleotide either does not appear at all or appears exactly twice (hence, the length of a DNA molecule is an even integer between 2 and 52). In case a nucleotide occurs twice, each occurrence can be of either type independently. Alien bacteria have two types of extremities, which in the technical biological jargon are referred to as arms and legs. A major discovery of Dr. Poucher's team is a method to determine the exact number of arms and legs of a bacterium by examining its DNA structure.

Here we represent each nucleotide as a letter of the alphabet. We refer to the different nucleotides as `a`, `A`, ... `z`, `Z`, where the lowercase and uppercase forms of a letter represent the two possible faces a nucleotide may appear with; we shall also use a/A, b/B, ... z/Z to refer to a nucleotide in either face.

To determine the number of extremities, Dr. Poucher starts by initializing two counters of arms and legs to zero, and then proceeds to perform a number of surgeries, transforming a DNA sequence into another one. After each transformation, you may need to increase some of the counters, depending on the type of surgery applied. When the empty sequence of nucleotides (which will be denoted by $\emptyset$) has been reached, the number of extremities of the original molecule has been found. The possible surgeries are:

1. Eliminate consecutive instances of a given nucleotide appearing with opposite faces. The number of arms and legs is preserved. For example: `aBbCaC` $\rightarrow$ `aCaC` by eliminating `Bb`. Another example: `DeHhEd` $\rightarrow$ `eHhE` by eliminating `dD`. Remember that DNA structure is circular, so in our representation as a string the last and first letters are connected.

2. Eliminate consecutive nucleotides appearing with the same face. Add one to the number of arms. For example: `BBcgCg` $\rightarrow$ `cgCg` by eliminating `BB`. Another example: `xabyyaBX` $\rightarrow$ `xabaBX` by eliminating `yy`.

3. Eliminate a sequence of four nucleotides formed by two different nucleotides that appear alternately where different occurrences of the same nucleotide have opposite faces. Add one to the number of legs. For example: `dcDCefFe` $\rightarrow$ `efFe`, by eliminating `dcDC`. Another example: `cmNMnC` $\rightarrow$ `cC` by eliminating `mNMn`.

4. Cut and paste, the most sophisticated procedure. First, a nucleotide is selected, for instance a/A, and the DNA sequence is chopped into two linear chains such that the nucleotide appears once in each of them.

   Second, if both occurrences of a/A are of the same face, one of the chains is "inverted" by reversing the sequence and changing the face of every nucleotide in the chain.

   Then, the chains are combined by concatenating the subsequence occurring before `a` with the sub-sequence occurring after `A`, and the subsequence occurring after `a` with the sub-sequence occurring before `A`.

   Finally, two new a/A nucleotides are added to close the chain into a circular shape. The face of the new nucleotides are the same if the original pair of nucleotides selected had the same face, and is different otherwise.

   Formally, suppose you select the nucleotide a/A, and further assume for the moment that it appears both times with the face `a` (`A`). The cut and paste surgery turns sequences of the form $S_1\mathtt{a}S_2S_3\mathtt{a}S_4$

(respectively $S_1\mathtt{A}S_2S_3\mathtt{A}S_4$) into $S_2\mathtt{a}S_1\bar{S}_3\mathtt{a}\bar{S}_4$ (respectively $S_2\mathtt{A}S_1\bar{S}_3\mathtt{A}\bar{S}_4$). On the other hand, if nucleotide a/A appears with its two different faces, the surgery turns sequences of the form $S_1\mathtt{a}S_2S_3\mathtt{A}S_4$ into $S_2\mathtt{a}S_1S_4\mathtt{A}S_3$. S_1, S_2, S_3 and S_4 are arbitrary sub-chains (possibly empty). In both cases the original circular chain was chopped into $S_1(a/A)S_2$ and $S_3(a/A)S_4$.

For example (see the figure below): starting with the sequence `BacDcAbD`, we can get chains `BacDc` and `AbD`. Then, merging at nucleotide a/A we get the sequence `cDca'BbDA'` where `a'` and `A'` represent the new a/A nucleotides. Here, $S_1 =$ `B`, $S_2 =$ `cDc`, $S_3 = \emptyset$ and $S_4 =$ `bD`.

Another example: take the same DNA sequence `BacDcAbD`, and cut to get the chains `DBac` and `DcAb`; paste nucleotide c/C (in this case you need to reverse one chain, for example `BaCd`) to get the sequence `cDBadcBa`. Here, $S_1 =$ `DBa`, $S_2 = \emptyset$, $S_3 =$ `D` and $S_4 =$ `Ab`.

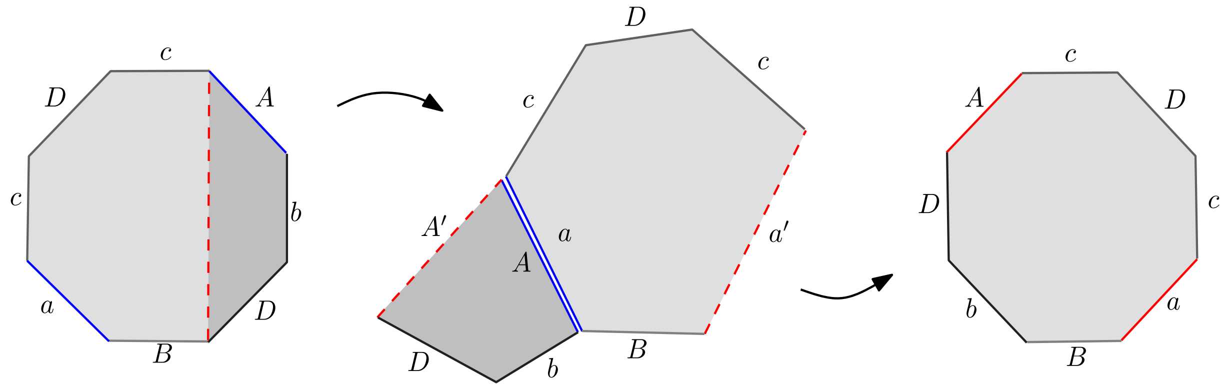


Figure 2: First example of use of cut&paste

This surgery does not modify the number of arms or legs, but can be used cleverly in combination with the previous surgeries to reduce the size of the DNA molecule and finish the calculation.

However, alien bacteria do not present both arms and legs at the same time. This is due to the fact that, in their early development, a leg, in the presence of one or more arms, becomes two arms. Because of the above, the end result is either a number of arms or a number of legs, but not both at the same time. In order to avoid expensive surgical procedures, Dr. Poucher has hired you to write a program that computes the number of arms and legs a bacterium will develop, given its DNA sequence. It is guaranteed that the result is determined uniquely by the original string, regardless of the particular sequence of surgeries applied.

Input

Each test case consists of a string of even length between 2 and 52, inclusive, representing the DNA structure of an alien bacterium. All characters are letters. There will be one case per line in the input. The last line contains the word "END" and must not be processed.

Output

The output for each test case should have exactly one line, containing the number of arms or legs the bacterium will have, followed by the word "arms" or "legs" respectively (if the number is 1, the words should be in singular). In case there will be neither arms nor legs, the program should print the word "none".

Sample Input

```
rkrk
abcdeABCDE
shcoOCfFHS
END
```

Sample Output

```
1 arm
2 legs
none
```

F
Haunted Graveyard

Tonight is Halloween and Scared John and his friends have decided to do something fun to celebrate the occasion: crossing the graveyard. Although Scared John does not find this fun at all, he finally agreed to join them in their adventure. Once at the entrance, the friends have begun to cross the graveyard one by one, and now it is the time for Scared John. He still remembers the tales his grandmother told him when he was a child. She told him that, on Halloween night, "haunted holes" appear in the graveyard. These are not usual holes, but they transport people who fall inside to some point in the graveyard, possibly far away. But the scariest feature of these holes is that they allow one to travel in time as well as in space; i.e., if you fall inside a "haunted hole", you appear somewhere in the graveyard a certain time before (or after) you entered the hole, in a parallel universe otherwise identical to ours.

The graveyard is organized as a grid of $W \times H$ cells, with the entrance in the cell at position $(0, 0)$ and the exit at $(W - 1, H - 1)$. Despite the darkness, Scared John can always recognize the exit, and he will leave as soon as he reaches it, determined never to set foot anywhere in the graveyard again. On his way to the exit, he can walk from one cell to an adjacent one, and he can only head to the North, East, South or West. In each cell there can be either one gravestone, one "haunted hole", or grass:

- If the cell contains a gravestone, you cannot walk over it, because gravestones are too high to climb.
- If the cell contains a "haunted hole" and you walk over it, you will appear somewhere in the graveyard at a possibly different moment in time. The time difference depends on the particular "haunted hole" you fell into, and can be positive, negative or zero.
- Otherwise, the cell has only grass, and you can walk freely over it.

He is terrified, so he wants to cross the graveyard as quickly as possible. And that is the reason why he has phoned you, a renowned programmer. He wants you to write a program that, given the description of the graveyard, computes the minimum time needed to go from the entrance to the exit. Scared John accepts using "haunted holes" if they permit him to cross the graveyard quicker, but he is frightened to death of the possibility of getting lost and being able to travel back in time indefinitely using the holes, so your program must report these situations.

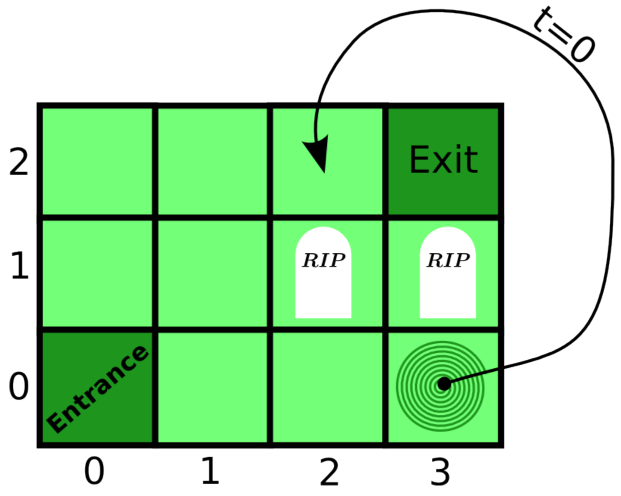


Figure 3: Sample graveyard

Figure **??** illustrates a possible graveyard (the second test case from the sample input). In this case there are two gravestones in cells $(2,1)$ and $(3,1)$, and a "haunted hole" from cell $(3,0)$ to cell $(2,2)$ with a difference in time of 0 seconds. The minimum time to cross the graveyard is 4 seconds, corresponding to the path:

$$(0,0) \rightarrow^{East}_{1\ sec} (1,0) \rightarrow^{East}_{1\ sec} (2,0) \rightarrow^{East}_{1\ sec} (3,0) \rightarrow^{hole}_{0\ sec} (2,2) \rightarrow^{East}_{1\ sec} (3,2)$$

If you do not use the "haunted hole", you need at least 5 seconds.

Input

The input consists of several test cases. Each test case begins with a line containing two integers W and H ($1 \leq W, H \leq 30$). These integers represent the width W and height H of the graveyard. The next line contains an integer G ($G \geq 0$), the number of gravestones in the graveyard, and is followed by G lines containing the positions of the gravestones. Each position is given by two integers X and Y ($0 \leq X < W$ and $0 \leq Y < H$).

The next line contains an integer E ($E \geq 0$), the number of "haunted holes", and is followed by E lines. Each of these contains five integers $X1, Y1, X2, Y2, T$. $(X1, Y1)$ is the position of the "haunted hole" ($0 \leq X1 < W$ and $0 \leq Y1 < H$). $(X2, Y2)$ is the destination of the "haunted hole" ($0 \leq X2 < W$ and $0 \leq Y2 < H$). Note that the origin and the destination of a "haunted hole" can be identical. T ($-10\,000 \leq T \leq 10\,000$) is the difference in seconds between the moment somebody enters the "haunted hole" and the moment he appears in the destination position; a positive number indicates that he reaches the destination after entering the hole. You can safely assume that there are no two "haunted holes" with the same origin, and the destination cell of a "haunted hole" does not contain a gravestone. Furthermore, there are neither gravestones nor "haunted holes" at positions (0,0) and (W-1,H-1).

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

Output

For each test case, if it is possible for Scared John to travel back in time indefinitely, output `Never`. Otherwise, print the minimum time in seconds that it takes him to cross the graveyard from the entrance to the exit if it is reachable, and `Impossible` if not.

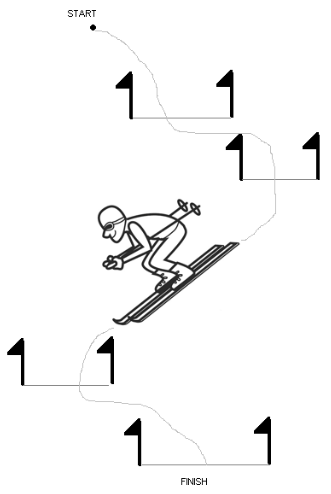
Sample Input

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

Sample Output

```
Impossible
4
Never
```

G
Slalom



In spite of the scarcity of snowfall in Madrid, interest in winter sports is growing in the city, especially with regard to skiing. Many people spend several weekends or even full weeks improving their skills in the mountains.

In this problem we deal with only one of the multiple alpine skiing disciplines: slalom. A course is constructed by laying out a series of gates, which are formed by two poles. The skier must pass between the two poles forming each gate. The winner is the skier who takes the least time to complete the course while not missing any of the gates.

You have recently started to learn to ski, but you have already set yourself the goal of taking part in the Winter Olympic Games of 2018, for which Madrid will presumably present a candidature. As part of the theoretical training, you need to write a program that calculates, given a starting point and a series of gates, the minimum-length path starting from the point given and passing through each gate until you reach the last one, which is the finish line. You may assume that the gates are horizontal and are ordered from highest to lowest, so that you need to pass through them in order. You consider yourself an accomplished skier, so you can make any series of turns, no matter how difficult, and your only concern is minimizing the total length of the path.

Input

The first line of each case gives the number of gates n ($1 \leq n \leq 1\,000$). The next line contains two floating point numbers, the Cartesian coordinates x and y of the starting position, in that order. Next come n lines with three floating point numbers each, y x_1 x_2, meaning that the next gate is a horizontal line from (x_1, y) to (x_2, y). You can safely assume that $x_1 < x_2$. The values of y are strictly decreasing and are always smaller than that of the starting position. The last gate represents the finish line. All coordinates are between $-500\,000$ and $500\,000$, inclusive. A value of 0 for n means the end of the input. A blank line follows each case.

Output

For each test case, output a line with the minimum distance needed to reach the finish line. Your answer should be accurate to within an absolute or relative error of 10^{-7}.

Sample Input

```
2
0 2
1 1 2
0 0.5 3

3
0 4
3 1 2
2 -1 0
1 1 2

0
```

Sample Output

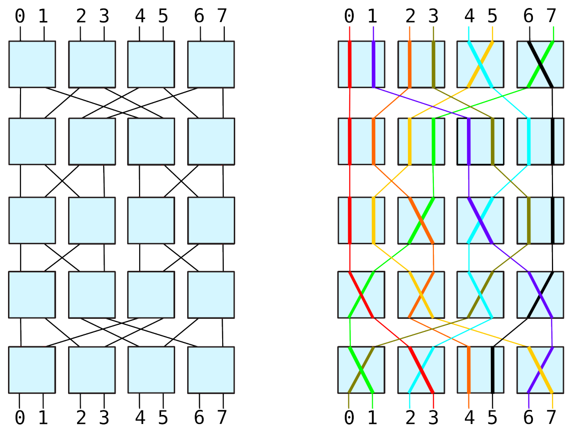
```
2.41421356237
4.24264068712
```

H
Routing

You work as an engineer for the *Inane Collaboration for Performance Computing*, where you are in charge of designing an intercommunication network for their computers. The network is arranged as a rectangular array of $2n-1$ rows, each having 2^{n-1} switches. A switch is a device with two input wires, X and Y, and two output wires, X' and Y'. If the switch is off, data from input X will be relayed to output X', and data from Y to Y'. If it is on, X will be connected to Y' and Y to X'. Additionally, there are 2^n computers in the topmost and bottommost rows, and messages need to be sent between pairs of them. Notice that data from two different sources cannot share a wire but, of course, both pieces of data can be routed through the same switch on different inputs.

You have come to the conclusion that the network that best suits your purposes has the Beneš topology. A 1-Beneš network is just a switch. For $n > 1$, a n-Beneš network can be constructed recursively as follows:

- In the first (top) row there are 2^{n-1} switches such that switch j $(0 \le j < 2^{n-1})$ has data inputs from computers $2j$ and $2j+1$ (we label the computers in the topmost and bottommost rows with integers between 0 and $2^n - 1$, inclusive, from left to right).
- Then a *perfect shuffle* permutation is applied to the output wires between the first and the second rows of switches, meaning that output number j in a row is connected to input number j' in the next row, where j' is obtained by rotating the n-bit pattern representing j in binary one bit to the right (again, inputs and outputs are numbered from left to right).
- If $n > 2$, the next rows of switches, up to (and including) the last-but-one, form two $(n-1)$-Beneš subnetworks, one on the left side and the other on the right side.
- Finally, the *inverse* shuffle permutation is applied to the outputs and a last row of switches is added.



(a) Topology

(b) Switch states for second sample input

Figure 4: 3-Benes network

For example, Figure **??** shows the Beneš network for $n = 3$ (squares represent switches; computers in the top and bottom rows are not drawn, but assigned with integers from 0 to 7). Figure **??** shows a possible state of the switches; squares where two of the lines cross are switches that have been turned on. You may verify that this state allows us to simultaneously establish communication paths from computers $0, 1, 2, 3, 4, 5, 6, 7$ at the bottom to $3, 7, 4, 0, 2, 6, 1, 5$ at the top, respectively.

You are given a set of pairs (a, b) of computers to connect simultaneously (where a is a computer in the bottom row and b a computer in the top row) by means of wire-disjoint paths, and you are to find how to select the state of all switches so that this can be accomplished.

Input

The first line of each test case is an integer n ($1 \leq n \leq 13$), meaning that you have 2^n pairs of computers to connect, as described above. A line with $n = 0$ marks the end of the input and should not be processed.

Each line with $n > 0$ will be followed by another line containing 2^n integers. The i-th integer ($0 \leq i < 2^n$) will be the computer in the topmost row that the i-th computer in the bottommost row needs to communicate with.

Output

The output for each case should have $2n-1$ lines, each containing a binary string of length 2^{n-1} indicating, for each switch, whether it must be turned on (1) or off (0).

The input given will always have at least one solution. In case of several solutions, return the lexicographically smallest one. That is, the string in the top row must be lexicographically smallest; in case of a tie, the string in the second row must be lexicographically smallest, and so on.

Outputs for different test cases should be separated by a blank line.

Sample Input

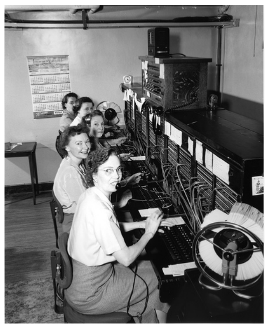
```
2
3 2 1 0
3
3 7 4 0 2 6 1 5
0
```

Sample Output

```
00
11
11

0011
0000
0110
1111
1101
```

I
Happy Telephones


Telephone operators. Photo: Seattle Municipal Archives.

In the land of Eden, all phone conversations are happy ones. People complaining on the phone are immediately put in jail. To enforce this law, the police taps all phone conversations.

The police wants to hire the approriate number of operators to listen to all conversations in a given period of time. Unfortunately, each of their operators can listen to one conversation only before needing a really long break to rest from the effort.

As a contractor of the police department, you have been asked to provide a program capable of determining the required number of operators. If the program does not work correctly, you will be put in jail as well, along with all the unhappy complainers. Do you really want to end up there?

Input

Each test case starts with two integers denoting the number of phone calls N ($1 \leq N < 10\,000$) and the number of intervals M ($1 \leq M < 100$). This is followed by N lines describing the telephone calls, each one consisting of four integers *Source*, *Destination*, *Start* and *Duration*. *Source* and *Destination* identify the pair of telephone numbers establishing the connection ($0 \leq Source, Destination \leq 10\,000\,000$). *Start* and *Duration* are the start time and duration of the call in seconds ($1 \leq Duration \leq 10\,000$ and $Start \geq 0$). You can safely assume that the sum of *Start* and *Duration* fits into a 32-bit signed integer.

Afterwards follow M lines containing the time intervals the police are interested in, each on described by two integers *Start* and *Duration*, in the same format and with the same meaning and constraints as those in the telephone calls. The last test case is represented by $N = M = 0$ and must not be processed.

Output

For each of the M intervals of each test case, print the number of calls that are active during at least one second of the interval.

Sample Input

```
3 2
3 4 2 5
1 2 0 10
6 5 5 8
0 6
8 2
1 2
8 9 0 10
9 1
10 1
0 0
```

Sample Output

```
3
2
1
0
```

J
Stammering Aliens

Dr. Ellie Arroway has established contact with an extraterrestrial civilization. However, all efforts to decode their messages have failed so far because, as luck would have it, they have stumbled upon a race of stuttering aliens! Her team has found out that, in every long enough message, the most important words appear repeated a certain number of times as a sequence of consecutive characters, even in the middle of other words. Furthermore, sometimes they use contractions in an obscure manner. For example, if they need to say *bab* twice, they might just send the message *babab*, which has been abbreviated because the second *b* of the first word can be reused as the first *b* of the second one.

Thus, the message contains possibly overlapping repetitions of the same words over and over again. As a result, Ellie turns to you, S.R. Hadden, for help in identifying the gist of the message.

Given an integer m, and a string s, representing the message, your task is to find the longest substring of s that appears at least m times. For example, in the message *baaaababababbababbab*, the length-5 word *babab* is contained 3 times, namely at positions 5, 7 and 12 (where indices start at zero). No substring appearing 3 or more times is longer (see the first example from the sample input). On the other hand, no substring appears 11 times or more (see example 2).

In case there are several solutions, the substring with the rightmost occurrence is preferred (see example 3).

Input

The input contains several test cases. Each test case consists of a line with an integer m ($m \geq 1$), the minimum number of repetitions, followed by a line containing a string s of length between m and 40 000, inclusive. All characters in s are lowercase characters from "a" to "z". The last test case is denoted by $m = 0$ and must not be processed.

Output

Print one line of output for each test case. If there is no solution, output `none`; otherwise, print two integers in a line, separated by a space. The first integer denotes the maximum length of a substring appearing at least m times; the second integer gives the rightmost starting position of this substring.

Sample Input

```
3
baaaababababbababbab
11
baaaababababbababbab
3
cccccc
0
```

Sample Output

```
5 12
none
4 2
```