Problem A. "Divisibility"

Consider an arbitrary sequence of integers. One can place + or - operators between integers in the sequence, thus deriving different arithmetical expressions that evaluate to different values. Let us, for example, take the sequence: 17, 5, -21, 15. There are eight possible expressions:

17 + 5 + -21 + 15 = 16
17 + 5 + -21 - 15 = -14
17 + 5 - -21 + 15 = 58
17 + 5 - -21 - 15 = 28
17 - 5 + -21 + 15 = 6
17 - 5 + -21 - 15 = -24
17 - 5 - -21 + 15 = 48
17 - 5 - -21 - 15 = 18

We call the sequence of integers **divisible** by *K* if + or - operators can be placed between integers in the sequence in such way that resulting value is divisible by *K*. In the above example, the sequence is divisible by 7 (17+5+-21-15=-14) but is not divisible by 5.

You are to write a program that will determine divisibility of sequence of integers.

Input

The first line of the input file contains two integers, *N* and *K* ($1 \le N \le 10000$, $2 \le K \le 100$) separated by a space.

The second line contains a sequence of *N* integers separated by spaces. Each integer is not greater than 10000 by it's absolute value.

Output

Write to the output file the word "Divisible" if given sequence of integers is divisible by *K* or "Not divisible" if it's not.

Sample input #1	**Sample input #2**
`4 7` `17 5 -21 15`	`4 5` `17 5 -21 15`
Output for the sample input #1	**Output for the sample input #2**
`Divisible`	`Not divisible`

Problem B. "Loan"

Mr. Smith received a loan for Q dollars. He plans to pay off this loan in K years at an interest rate of P percent per year. That means that, after each year, Mr. Smith's debt grows by $P*Q'/100$ dollars (Q' being the debt at the beginning of that year) and his annual payment is deducted from his debt.

For the first year, Mr. Smith wants to pay minimal amount that will allow him to pay off the loan within exactly K years. For each subsequent year, he is willing to pay either the same amount as the previous year or one cent less than the previous year's payment. He wants the loan to be completely paid off without overpayment of even a single cent by the end of the K^{th} year.

The bank performs all transactions with a precision of one cent, and calculates interest due at the end of each year. Whenever interest is calculated, the result is immediately rounded to the nearest cent, with 0.5 cents rounded up.

Input

The input file contains a single line with three numbers Q, P and K, separated by spaces. Q is a real number ($10 \le Q \le 1000000$) given with no more than two digits to the right of the decimal point. This value represents the amount of the loan in dollars. One one-hundredth of a dollar is a cent. $P$and K are integers ($0 \le P \le 100$, $1 \le K \le 100$).

Output

Write to the output file a schedule of payments for Mr. Smith. You should write the amount of each payment and number of years that payment should be made, thus effectively grouping equal payments. Each group of equal payments must be written on separate line, with no blank lines between them. The output format for each group of payments is:

```
$X for Y year(s)
```

where X is payment amount in dollars printed with exactly two digits after decimal point. Y is a number of years for which this payment should be made. The dollar value given on each line must be one cent less than the dollar value printed above it.

If there are multiple correct payment schedules, you can output any one of them, but the first payment should be the minimal possible one. If no solution can be found for the given input, then the output file shall contain only the word "Impossible".

Sample input #1

```
100 100 100
```

Output for the sample input #1

```
Impossible
```

Sample input #2

```
20 0 10
```

Output for the sample input #2

```
$2.00 for 10 year(s)
```

Sample input #3

```
939850.83 85 35
```

Sample output for the sample input #3

```
$798873.22 for 1 year(s)
$798873.21 for 1 year(s)
$798873.20 for 1 year(s)
$798873.19 for 1 year(s)
$798873.18 for 1 year(s)
$798873.17 for 4 year(s)
$798873.16 for 1 year(s)
$798873.15 for 2 year(s)
$798873.14 for 1 year(s)
$798873.13 for 3 year(s)
$798873.12 for 1 year(s)
$798873.11 for 4 year(s)
$798873.10 for 2 year(s)
$798873.09 for 7 year(s)
$798873.08 for 2 year(s)
$798873.07 for 1 year(s)
$798873.06 for 1 year(s)
$798873.05 for 1 year(s)
```

Problem C. "Expression"

It is known that Sheffer stroke function (NOT-AND) can be used to construct any Boolean function. The truth table for this function is given below:

Truth table for Sheffer stroke function

x	**y**	**x|y**
0	0	1
0	1	1
1	0	1
1	1	0

Consider the problem of adding two binary numbers *A* and *B*, each containing *N* bits. The individual bits of *A* and *B* are numbered from 0 (the least significant) to *N*-1 (the most significant). The sum of *A* and *B* can always be represented by *N*+1 bits. Let's call most significant bit of the sum (bit number *N*) the **overflow** bit.

Your task is to construct a logical expression using the Sheffer stroke function that computes the value of the overflow bit for arbitrary values of *A* and *B*. Your expression shall be constructed according to the following rules:

1. `Ai` is an expression that denotes value of i^{th} bit of number *A*.
2. `Bi` is an expression that denotes value of i^{th} bit of number *B*.
3. `(`*x*`|`*y*`)` is an expression that denotes the result of Sheffer stroke function for *x* and *y*, where *x* and *y* are expressions.

When writing the index, *i*, for bits in *A* and *B*, the index shall be written as a decimal number without leading zeros. For example, bit number 12 of *A* must be written as `A12`. The expression should be completely parenthesized (according to the 3rd rule). No blanks are allowed inside the expression.

Input

The input file contains a single integer *N* ($1 \le N \le 100$).

Output

Write to the output file an expression for calculating overflow bit of the addition of two *N*-bit numbers *A* and *B* according to the rules given in the problem statement. The stroke symbol (|) is an ASCII character with code 124 (decimal).

The output file size shall not exceed 50**N* bytes.

Sample input

```
2
```

Sample output for the sample input

```
((A1|B1)|(((A0|B0)|(A0|B0))|((A1|A1)|(B1|B1))))
```

Problem D. "Computer Dialogue"

There are three computers connected by a network. One of them is a server and the other two are clients. The server has some files, each with a different name. The full name of each file consists of two parts, a name and an extension. Both clients know the full names of all of the files stored on the server. From among its files, the server chooses a single file and sends the name part of that file's full name to one of the clients and the extension part of the full name to the other client.

The clients then begin communicating in an effort to determine which file was selected by the server (they want to learn the file's full name). However, the clients have to communicate in a very restricted manner. Clients take turns sending messages to each other, but they can only say when they don't know the full name of the file. If one client does not know the full name of the chosen file, that client may send a message saying "I don't know the full file name" to the other client. The two clients alternate, sending only this message back and forth. This continues until either one of the clients knows the full file name or they decide to quit. The client that received the name part of the full file name always waits for the other client to send the first message.

Pretend that you know all the full file names that reside on the server (both the name and the extension part) and you are listening to the conversation between the clients. Based on this conversation, you should determine the set of files that might have been chosen by the server. Files in this set are called **candidate files**.

Input

The first line of the input file contains two integers N and M, separated by a space. N ($1 \le N \le 1000$) is the number of files of the server, and M ($1 \le M \le 100$) is the number of messages exchanged between the clients in their effort to determine the full file name.

Each of the next N lines contain one full file name. Full file names are given in a manner similar to MS-DOS 8.3 format. Each full file name is represented in name.extension form, where both the name and the extension consist of only capital, alphabetic characters and decimal digits. The name part will have at least one character and at most eight. The extension part will have at most three characters and may be empty. If extension is empty then separating dot may be omitted.

Each full file name appears in the input file no more than once.

Output

Write on the first line of the output file the number of candidate files based on the given set of files and the number of messages exchanged between the clients. Write zero if there are no candidate files.

On the following lines, write the full names of all candidate files. Each of these names should be written on a separate line. They should appear in the same order and with exactly the same spelling as in the input file. That means that if the separating dot was omitted in the input for a particular file then it should also be omitted for this file in the output and vice versa. No file may be listed more than once.

Sample input

```
19 2
LICENCE.TMP
WIN32.LOG
FILEID.
PSTOTEXT.TXT
GSVIEW32.EXE
GSVIEW32.ICO
GSVIEWDE.HLP
LICENCE
GSVIEWEN.HLP
GSVW32DE.DLL
FILEID.TMP
GSVW32EN.DLL
PSTOTXT3.DLL
PSTOTXT3.EXE
GSV16SPL.EXE
GVWGS32.EXE
ZLIB32.DLL
PRINTER.INI
README.TXT
```

Output for the sample input

```
6
LICENCE.TMP
FILEID.
LICENCE
FILEID.TMP
PSTOTXT3.DLL
PSTOTXT3.EXE
```

Problem E. "Lock Manager"

You are invited to be a part of the team that is developing yet another DBMS (Data Base Management System). You will be responsible for the Lock Manager.

Locks control concurrent access to data items by multiple transactions. Your DBMS is simple and uses only Shared (S) and Exclusive (X) mode locks. Each lock request contains a lock mode (S or X), a transaction identifier and a data item identifier. Multiple locks can be granted to the same data item as long as none of them **conflict**.

Two locks for the same data item conflict if:

- they belong to different transactions, and
- at least one of them is exclusive (X) mode lock.

At the earliest stages of development you are asked to write very simple lock manager that processes lock requests. The lock is granted if it does not conflict with previously granted locks for this data item. Your task is simple: locks, once granted, are never released or changed in any way. If lock request is denied due to conflict with some previously granted lock, then transaction making this request is blocked and all further requests from this transaction are ignored.

Input

The input file consists of a number of lock requests, each request on a different line. Requests have the following format:

```
MODE TRID ITEM
```

Where MODE is a single capital letter S or X denoting requested lock mode. TRID and ITEM are transaction identifier and data item identifier correspondingly. Both TRID and ITEM are integers, both are greater than zero, and both consist of at most 9 decimal digits.

There are at least one and at most 10000 requests in the input file.

The last request is followed by a line consisting of a single character '#'.

Output

Your program shall sequentially process all requests from the input file. For each request you should write one line that contains the response to the request. The following responses are allowed:

- GRANTED - the lock request does not conflict with any previously granted locks and is granted.
- DENIED - the lock request conflicts with some previously granted lock and is denied, thus blocking the requesting transaction.
- IGNORED - the transaction was blocked on some request before this one.

Responses shall appear in all capital letters exactly as shown above. An arbitrary number of blank lines can follow last response in the output file.

Sample input

```
S 1 1
S 2 2
X 10 1
S 6 123456789
S 3 3
X 2 2
S 5 6
S 3 1
S 3 2
X 987654321 123456789
X 1 4
S 6 6
S 3 5
S 2 4
X 4 5
S 2 51
#
```

Output for the sample input

```
GRANTED
GRANTED
DENIED
GRANTED
GRANTED
GRANTED
GRANTED
GRANTED
DENIED
DENIED
GRANTED
GRANTED
IGNORED
DENIED
GRANTED
IGNORED
```

Problem F. "Dictionary"

Authors of the new, all-in-one encyclopedia have organized the titles in the order they consider most appropriate for their readers. It's not always alphabetical, because they want to observe some peculiar relationships between them. However, they still want to allow users to look up titles quickly.

They achieve this by adding a carefully calculated number of spaces before every title in the list of titles. They call this structure **a dictionary**.

A dictionary is represented by a list of words with some number of spaces before certain words. Dictionary format can be described as a set of constraints on sequences of consecutive words starting with the same letter. Any maximal sequence of consecutive words starting with the same letter should satisfy the following rules:

- The first word in the group has no spaces before it. Every subsequent word in the group has at least one leading space.
- If
    - the first word of the group is deleted and
    - one space is deleted before every remaining word and
    - the first letter is deleted from every remaining word

  then resulting sequence is a dictionary.

The authors don't feel like giving you a more detailed explanation of what a dictionary is, so they have included an example (see sample input and output) that clarifies their definition.

Your task is to write a program that will convert a given list of words into a dictionary by adding some number of spaces before certain words and preserving the original order of the words.

Input

The input file consists of at least one and most 100000 words. Each word consists of at least one and at most 10 lower-case letters. There will be no leading or trailing spaces. There will be no blank lines between the words, but there may be an arbitrary number of blank lines at the end of the file.

Output

Write to the output file the original words in the same order without any trailing spaces but with the appropriate number of leading spaces, so that this word list is a dictionary. There should be no blank lines between the words, but there may be an arbitrary number of blank lines at the end of the file.

Sample input

```
a
ant
antique
amaze
bargain
bridge
bride
bribe
born
```

```
bucket
tart
tan
tram
trolley
t
try
trial
zed
double
dorm
do
dormant
donate
again
agony
boost
back
born
```

Sample output for the sample input

NOTE: *For reading convenience spaces are replaced with* `'.'` *characters. Your output file should contain spaces instead.*

```
a
.ant
..antique
.amaze
bargain
.bridge
..bride
...bribe
.born
.bucket
tart
.tan
.tram
..trolley
.t
.try
..trial
zed
double
.dorm
..do
..dormant
..donate
again
.agony
boost
.back
.born
```

Problem G. "Highways"

The island nation of Flatopia is perfectly flat. Unfortunately, Flatopia has a very poor system of public highways. The Flatopian government is aware of this problem and has already constructed a number of highways connecting some of the most important towns. However, there are still some towns that you can't reach via a highway. It is necessary to build more highways so that it will be possible to drive between any pair of towns without leaving the highway system.

Flatopian towns are numbered from 1 to *N* and town *i* has a position given by the Cartesian coordinates (x_i, y_i). Each highway connects exactly two towns. All highways (both the original ones and the ones that are to be built) follow straight lines, and thus their length is equal to Cartesian distance between towns. All highways can be used in both directions. Highways can freely cross each other, but a driver can only switch between highways at a town that is located at the end of both highways.

The Flatopian government wants to minimize the cost of building new highways. However, they want to guarantee that every town is highway-reachable from every other town. Since Flatopia is so flat, the cost of a highway is always proportional to its length. Thus, the least expensive highway system will be the one that minimizes the total highways length.

Input

The input file consists of two parts. The first part describes all towns in the country, and the second part describes all of the highways that have already been built.

The first line of the input file contains a single integer N ($1 \le N \le 10000$), representing the number of towns. The next N lines each contain two integers, x_i and y_i separated by a space. These values give the coordinates of i^{th} town (for i from 1 to N). Coordinates will have an absolute value no greater than 10000. Every town has a unique location.

The next line contains a single integer M ($0 \le M \le 10000$), representing the number of existing highways. The next M lines each contain a pair of integers separated by a space. These two integers give a pair of town numbers which are already connected by a highway. Each pair of towns is connected by at most one highway.

Output

Write to the output file a single line for each new highway that should be built in order to connect all towns with minimal possible total length of new highways. Each highway should be presented by printing town numbers that this highway connects, separated by a space. If no new highways need to be built (all towns are already connected), then the output file should be created but it should be empty.

9 1 5 0 0 3 2 4 5 5 1 0 4 5 2 1 2 5 3 3 1 3 9 7 1 2	1 6 3 7 4 9 5 7 8 3

Problem H. "Advertisement"

The Department of Recreation has decided that it must be more profitable, and it wants to sell advertising space along a popular jogging path at a local park. They have built a number of billboards (special signs for advertisements) along the path and have decided to sell advertising space on these billboards. Billboards are situated evenly along the jogging path, and they are given consecutive integer numbers corresponding to their order along the path. At most one advertisement can be placed on each billboard.

A particular client wishes to purchase advertising space on these billboards but needs guarantees that every jogger will see it's advertisement at least K times while running along the path. However, different joggers run along different parts of the path. Interviews with joggers revealed that each of them has chosen a section of the path which he/she likes to run along every day. Since advertisers care only about billboards seen by joggers, each jogger's personal path can be identified by the sequence of billboards viewed during a run. Taking into account that billboards are numbered consecutively, it is sufficient to record the first and the last billboard numbers seen by each jogger.

Unfortunately, interviews with joggers also showed that some joggers don't run far enough to see K billboards. Some of them are in such bad shape that they get to see only one billboard (here, the first and last billboard numbers for their path will be identical). Since out-of-shape joggers won't get to see K billboards, the client requires that they see an advertisement on every billboard along their section of the path. Although this is not as good as them seeing K advertisements, this is the best that can be done and it's enough to satisfy the client.

In order to reduce advertising costs, the client hires you to figure out how to minimize the number of billboards they need to pay for and, at the same time, satisfy stated requirements.

Input

The first line of the input file contains two integers K and N ($1 \le K, N \le 1000$) separated by a space. K is the minimal number of advertisements that every jogger must see, and N is the total number of joggers. The following N lines describe the path of each jogger. Each line contains two integers A_i and B_i (both numbers are not greater than 10000 by absolute value). A_i represents the first billboard number seen by jogger number i and B_i gives the last billboard number seen by that jogger. During a run, jogger i will see billboards A_i, B_i and all billboards between them.

Output

On the fist line of the output file, write a single integer M. This number gives the minimal number of advertisements that should be placed on billboards in order to fulfill the client's requirements. Then write M lines with one number on each line. These numbers give (in ascending order) the billboard numbers on which the client's advertisements should be placed.

5 10 1 10 20 27 0 -3 15 15 8 2 7 30 -1 -10 27 20 2 9 14 21	19 -5 -4 -3 -2 -1 0 4 5 6 7 8 15 18 19 20 21 25 26 27

Problem I. "Random Route"

Where do you want to go today, and how do you want to get there? You decide to choose the answer to both questions at random.

You will be given a list of roads. Each road connects one city to another city (all roads are one-way), and each road takes a certain amount of time to drive. You will also be given a starting city. Consider all cities that you're able to drive to, not including your starting city, and choose one of them at random with uniform probability to be your destination. Now consider every fastest route from your starting city to your destination city, and choose one of these routes at random with uniform probability. This will be the route on which you end up driving.

For each road in the input, your program must output the probability that you will end up driving on that road, given the behavior outlined above.

Input

The first line of input gives the number of cases, N ($1 \leq N \leq 100$). N test cases follow. Each case begins with a line formatted as: `num_roads starting_city`

This will be followed by num_roads ($2 \leq$ num_roads ≤ 50) lines, each formatted as: `city1 city2 time`

Each line represents a one-way road that starts at city1 and ends at city2, and takes time hours to drive. All cities will be formatted as strings consisting of only lowercase letters and underscores. For each road, city1 will not be equal to city2, and time will be an integer between 1 and 100000, inclusive. The starting city is guaranteed to appear as city1 on at least one road; therefore, there will always be at least one possible destination (and at least one shortest route to that destination).

Output

For each test case, output one line containing "Case #x: " followed by the probability that you will drive on each road, in the same order that the roads were listed in the input. Probabilities should be space separated and formatted so there are exactly seven digits after the decimal point. Each probability must be within a distance of 1e-6 from the correct answer to be judged as correct.

1 5 san_francisco san_francisco los_angeles 6 los_angeles san_diego 2 san_francisco san_diego 8 los_angeles san_diego 2 san_francisco los_angeles 6	Case #1: 0.4500000 0.2000000 0.1000000 0.2000000 0.4500000

Problem J. "Hexagon Game"

You are playing a game on a hexagonal board of size S. The middle row is composed of S hexagons, and the top and bottom rows each have (S + 1) / 2 hexagons. (S will be odd.) The hexagons are numbered starting with 1 in the upper left, and increasing left-to-right and top-to-bottom. Here is a hexagonal board of size S=5.

The game starts with S checkers on the board. Multiple checkers might start in the same position. Each checker also has an associated integer value between 0 and 50, inclusive. A turn consists of choosing a checker and moving it to an adjacent position, which increments your score by the value of that checker. Checkers cannot move off the board. Each position can contain any number of checkers at the same time.

The game ends when all the checkers are lined up in a straight row, with exactly one checker per hexagon. There are three possible ending configurations on any board. For S=5, the game will end when checkers are in positions (1, 5, 10, 15, 19), or in positions (3, 6, 10, 14, 17), or in positions (8, 9, 10, 11, 12). Your program must output the smallest possible score of a finished game.

For example, assume the checkers start in positions (1, 2, 5, 15, 19). The checker in position 1 has a value of 1, the checkers in positions 2 and 5 have values of 3, and the checkers in positions 15 and 19 have values of 0. You could move the checker from position 1 into position 5 and then position 10. Both of these moves add one point to your score. Then you could move the checker from position 2 into position 1, adding three points to your score. This game would end with a score of 5, which is the lowest possible score for this starting configuration.

Input

The first line of input gives the number of cases, N ($1 \leq N \leq 100$). N test cases follow. Each case consists of two lines. The first line contains the starting positions of the checkers, space separated. The second line contains the values of each checker, respectively, space separated. It is guaranteed that $3 \leq S \leq 75$, S is odd.

Output

For each test case, output one line containing "Case #x: " followed by the minimum possible score of a finished game.

1 1 2 5 15 19 1 3 3 0 0	Case #1: 5

Problem K. "Completion"

You are in a chat room that supports nickname tab-completion. Suppose the input buffer now contains the string *s* and you decide to use the completion facility. The first time you press tab you will be shown the lexicographically first element of the options that has *s* as a prefix. Pressing tab again will give the lexicographically second, and so forth. Once the possible options are exhausted the tab key will do nothing. Having found a completion that suits you, you may either press enter to complete the word you are typing, or continue typing characters into the input buffer. If you decide to type characters, they will be appended to the current completion. Your goal is to type the word *w*, followed by the enter key, using as few keystrokes as possible. Each character and each tab key count as single keystrokes. By interchanging character typing and tab completion sequences as many times as you like, return the fewest number of keystrokes required.

In order to reduce advertising costs, the client hires you to figure out how to minimize the number of billboards they need to pay for and, at the same time, satisfy stated requirements.

Input

The first line of the input contains integer n ($0 \le n \le 100$) - the number of options. The following n lines contain options, one option per line. Each option is a string containing only lowercase Latin letters, the length is between 1 and 100 inclusive. The last line contains the target string *w* containing only lowercase Latin letters, the length is between 1 and 100 inclusive.

Output

Output the required minimal number of keystrokes.

2 myn myname myname	3
0 myname	7
7 abc ab abcd frankies frank a a frankie	5

Problem L. "Cow Acrobats"

Farmer John's N ($1 \le N \le 50000$) cows (numbered 1..N) are planning to run away and join the circus. Their hoofed feet prevent them from tightrope walking and swinging from the trapeze (and their last attempt at firing a cow out of a cannon met with a dismal failure). Thus, they have decided to practice performing acrobatic stunts.

The cows aren't terribly creative and have only come up with one acrobatic stunt: standing on top of each other to form a vertical stack of some height. The cows are trying to figure out the order in which they should arrange themselves within this stack.

Each of the N cows has an associated weight ($1 \le W_i \le 10000$) and strength ($1 \le S_i \le 1{,}000{,}000{,}000$). The risk of a cow collapsing is equal to the combined weight of all cows on top of her (not including her own weight, of course) minus her strength (so that a stronger cow has a lower risk). Your task is to determine an ordering of the cows that minimizes the greatest risk of collapse for any of the cows.

Input

The first line contains the integer N. The following N lines describe cows, cow i is given as two space-separated integers, W_i and S_i.

Output

Print a single integer, giving the largest risk of all the cows in any optimal ordering that minimizes the risk.

3 10 3 2 5 3 3	2

Put the cow with weight 10 on the bottom. She will carry the other two cows, so the risk of her collapsing is 2+3-3=2. The other cows have lower risk of collapsing.