Problem A. Optimal bribing

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

It is not easy to live and do business in Russia. For example, to get approval for gas utilities installation according to Russian law you need to get project documentation from city project bureau. But to get project documentation you will be required to get approval for gas utilities installation. Even to people who don't write computer programs and don't know buzz words like «endless recursion» it is clear that to get access to gas one needs to make a wonder. For example, to give bribe or kickback.

Suppose two businessmen compete for some business project which brings them profit V. But only one businessman will be able to win the project — namely, the one who will get N approvals from bureaucrats faster than another. Every businessman gets approvals in certain order — he cannot file next document for approval while previous document is not approved. To speed the process of getting approval up, businessmen may bribe bureaucrats. A businessman may visit (and bribe) only one bureaucrat a day. Specifically, if in particular day businessman i gives to a bureaucrat bribe of b units (non-negative real number which businessman defines itself, and b may vary from day to day), probability that the bureaucrat will give him the approval at that day is $1 - 0.99(1 - F_i)^b$, where F_i is businessman-specific fascination parameter (so, you see that without a bribe there is only 1% chance to get an approval at a particular day). If bribing was unsuccessful, businessman may try again and again (and this doesn't change behavior of bureaucrat in any way), but he loses time (remember, businessman can give only one bribe a day) and money (since even in the case of unsuccessful bribe bureaucrat takes it). If a businessman gets an approval, he can file next document for approval on the next day (and at the same day bribe and get the document approved in the case of luck). Thus, even very lucky and fascinating businessman cannot have more than one document approved in one day. The first businessman who gets N approvals wins the project and makes profit V, another gets nothing (but loss from bribing). If businessmen get all necessary approvals at the same day, we consider that the winner is determined by coin tossing (that is, each of them has the same chance to win). Every businessman knows how many approvals the other one has at any moment.

Suppose that every businessman acts optimally to maximize expected difference between profit and costs (which is the total sum of bribes paid), knowing the strategy of another player. What is the probability to win the project for each of them?

Input

Input file contains numbers N, F_1, F_2, V from the problem statement. N and V are integers, F_1 and F_2 are given with not more than two digits after decimal point ($1 \leqslant N \leqslant 10$, $0 < F_1, F_2 < 1$, $1 \leqslant V \leqslant 100$). Numbers will be chosen so that businessmen will have unique equilibrium combination of optimal bribing strategies (so, required probabilities will be unique as well).

Output

Write to the output file required probabilities with accuracy not less than 10^{-6}.

Example

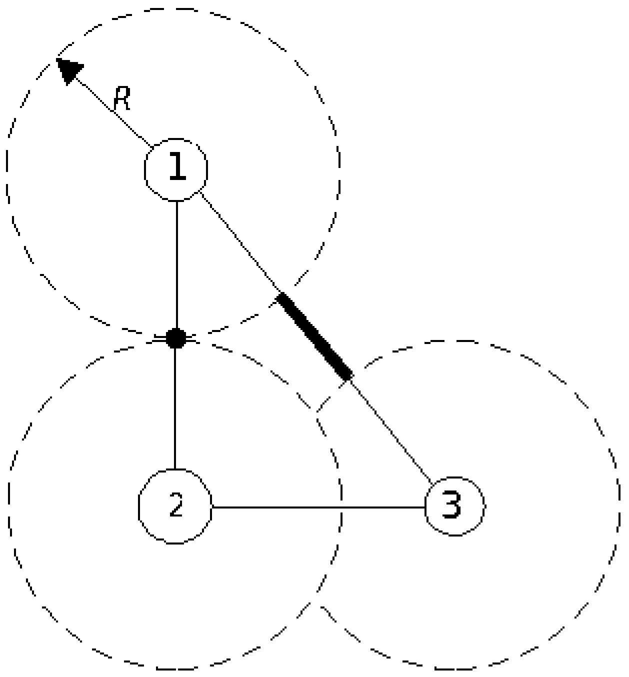
standard input	standard output
`1 0.14 0.10 20`	`0.618627007 0.381372993`

Problem B. Fire Station Building

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

There is a country with N cities connected by M bidirectional roads, and you need to build a fire station somewhere. Of course, your problem would be too simple without the following restrictions:

- Firemen should be able to reach any city from fire station by roads only.
- You want to minimize expected distance from fire station to place of fire. For this purpose, probability of a fire is given to you for every city. Assume that firemen always choose the shortest way to fire.
- You can place fire station not only in cities but on the roads between them as well. Moreover, sanity regulations of the country forbid placement of a fire station closer than at a distance R from cities. Distances are measured on roads only, so you can place fire station on a road if its length is not less than $2R$, and you should not worry about distances to cities not adjacent to the given road. In particular, $R = 0$ means that you are allowed to place a fire station in cities.



Example of a country with three cities and three roads. Places where you can build fire station are marked with bold line and bold dot.

You are given a complete description of the country. Find the best place for a fire station in it.

Input

The first line of input file contains integer numbers N, M and R ($1 \leqslant N \leqslant 100$, $0 \leqslant M \leqslant N(N-1)/2$, $0 \leqslant R \leqslant 10^4$). Second line contains N integer numbers — probabilities of a fire in each city. Numbers are non-negative integers given in hundredths of percent (that is, sum to 10^4). Each of the next M lines contains description of one road, namely three integer numbers A_i, B_i and L_i — endpoints of the road and its length in kilometers ($1 \leqslant A_i < B_i \leqslant N$, $1 \leqslant L_i \leqslant 10^4$). There can be at most one road between any two cities.

Output

Output only one number — expected length of a way to fire in kilometers assuming fire station is built optimally. This number should be precise up to 1 meter. If by some reason it is impossible to build fire station that fulfills all the requirements, write to the output file number −1 instead.

Examples

standard input	standard output

3 3 20
3000 5000 2000
1 2 40
1 3 50
2 3 30
	26.00000

3 1 20
3000 5000 2000
1 3 50
	-1

In the first example (which corresponds to the picture above) it is optimal to build fire station in the middle of the first road. In the second example it is impossible to build fire station satisfying all the requirements. In particular, any fire station on the map will violate requirement one (since the country is disconnected).

Problem C. Parking at Secret Object

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

A long time ago, in a Galaxy Far Far Away...

Imagine you develop fully automated parking system for Death S... well, secret military object of the Empire. The system manages N parking slots arranged along the equator of the Object and numbered from 1 to N in clockwise direction. Objective of the system is to answer queries of groups of battle dro... let us call them users that from time to time take off and get in to the Object. The only requirement that should be obeyed during processing the inquiries is that every group of users should get some set of unoccupied parking slots adjacent to each other. For example, suppose that the Object has 10 parking slots. If a group of five users calls for permission to get in, you may allot for them, for instance, parking slots from 2 to 6 or 1-2 and 8-10 (remember, parking slots are arranged along the equator of the Object, so they form a circle and slots 1 and N are the neighboring ones).

Let us define a *cluster* as a maximal (by inclusion) group of unoccupied neighboring slots and the *size* of a cluster as the number of slots in it. Correspondingly, define the *number* of a cluster as the number of the leftmost parking slot in the cluster (the first parking slot when you look over all parking slots of the cluster in clockwise direction), and call this parking slot the *head slot* of the cluster. If all parking slots in the system are unoccupied, then we treat it as one cluster consisting of all parking slots and having head slot number 1.

To improve efficiency of the parking system you decided to use the following algorithm for determining slots to allot for incoming users. Suppose a group of S users is coming in the land.

- You choose for them cluster of minimum size not less than S.
- If there is no such cluster, you reject the query.
- If there are several such clusters, you choose the one with minimum number.
- You allot for users S neighboring parking slots starting from head slot of the cluster and going in clockwise direction.

What is left is to implement the logic of the parking system efficiently.

Input

The first line of input file contains two integer numbers N and Q — number of parking slots in the system and number of queries of users ($1 \leqslant N, Q \leqslant 10^5$). The second line contains characters «`.`» and «`X`», which represent unoccupied and taken up slots in the system, respectively (starting from the first slot in clockwise direction). i-th character is indicator of whether i-th parking slot in the system occupied or not. The following Q lines contain queries of users. Every line represents i-th query and has one of the two forms:

`PARK` S_i — group of S_i users wants to land ($1 \leqslant S_i \leqslant N$).

`LEAVE` Q_i — group of users from query Q_i wants to take off ($1 \leqslant Q_i < i$, queries are numbered from 1 to Q in the order they appear in the input file).

All queries are consistent, so, for example, group of already flown away users cannot query for taking off, or «`LEAVE`» query cannot contain reference to another «`LEAVE`» query.

Output

For every «PARK» query in the input file output the only line containing description of set of parking slots allotted for corresponding group of users, or the message «NO ROOM» if it is impossible to meet corresponding request. In case of a positive answer description should be given in the format of ordered intervals precisely as in the examples provided to you below.

Example

standard input	standard output
10 4 .... PARK 4 PARK 3 LEAVE 1 PARK 4	1-4 5-7 1,8-10
10 11 ....X..X.. PARK 1 PARK 3 PARK 4 LEAVE 2 PARK 5 LEAVE 5 PARK 1 PARK 1 PARK 2 PARK 4 PARK 3	6 1,9-10 NO ROOM 1-3,9-10 7 9 1,10 NO ROOM 2-4

Problem D. Chessmaster

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

Ivan Petrovich and Petr Ivanovich enjoy playing chess, especially Ivan Petrovich. Each time he loses regular weekend game in chess, from superfluity of feelings he takes the board and breaks it into separate black and white fields. Well, things were going that way before Petr Ivanovich, frustrated by weekly breakages of his chessboards, replaced usual chessboard by titanic one. Now it was not so easy even to scratch it! But Ivan Petrovich didn't become flustered and, in affective state after usual unsuccessful play, ordered a powerful laser which could burn accurate perfectly round holes through the chessboard.

Only after the laser was delivered Ivan Petrovich realized to his horror that it was not powerful enough: instead of having diameter of a beam equal to the diagonal of a chessboard, his laser had diameter equal to the length of its side! This means that he will be unable to destroy the whole chessboard in one shot, and will have to use the laser several times. But Ivan Petrovich's pension is not large enough to cover bills for electricity after using the laser too frequently, so now he is puzzled with natural question: if he wishes to destroy at least $P\%$ of the chessboard surface, what is the minimum number of laser shots that he have to do?

Help Ivan Petrovich in answering this important and difficult question. And remember: you may shoot only in direction orthogonal to the surface of chessboard, and it is not allowed to move (probably) peeled off parts. Chessboard has the usual form of a perfect square.

Input

Input file contains up to 100 non-negative integer numbers, each on a separate line — percentage of the board P that Ivan Petrovich wants to destroy. Each P will not exceed 100, of course.

Output

For every P in the input file write to the output file on a separate line required minimum number of laser shots. Follow format shown in the example below.

Example

standard input	standard output

`1`
`2`
	`Case #1: 1`
	`Case #2: 1`

Problem E. A bit of classic

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

Everybody loves classical problems! Real professional can solve them quickly and gaily come to more difficult problems, and amateurs are just able to solve them, which is also important for popularization of computer programming contests. Here you are required to solve classical problem not only in chess but in computer science as well. Given integer N, find a way to bypass every cell of a chessboard $N \times N$ exactly once with a chess knight.

Input

Input file contains integer number N — size of the board ($1 \leqslant N \leqslant 250$).

Output

If there is no solution for the given size of the board, write to the output file the only message «`No solution.`» (without quotes). Otherwise, write to the first line of the output file message «`There is solution:`», and then to every of the N following lines write N numbers separated by spaces — order of traversal of the board. Each of the numbers from 1 to N^2 should occur in this sequence exactly once. Knight may start and end its trip at any cell of the chessboard.

Examples

standard input	standard output
3	No solution.
5	There is solution: 1 14 9 20 3 24 19 2 15 10 13 8 25 4 21 18 23 6 11 16 7 12 17 22 5

Problem F. Ghostbusters

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 megabytes

Now famous Ghostbusters want your help. As you probably know, all ghosts that are caught are put in asylum to stay there forever (since they are already dead and cannot be obliterated). The problem is that though usually ghosts can intersect and even lie inside each other, some new ghosts by yet unknown to Egon reasons are afraid of some old ghosts and cannot intersect with them. That's what your problem is — to find the maximum radius of a new ghost that can be put to the asylum in the worst case (i.e., when it is afraid of all other ghosts in the asylum). You task is simplified by the fact that all ghosts are in fact perfect solid spheres and stay at the same points forever. Also don't forget that new ghost cannot intersect borders of the asylum (but can touch them as well as other ghosts).

Input

The first line of the input file contains three integer numbers W, H and D — sizes of the asylum given in meters ($1 \leqslant W, H, D \leqslant 1000$). Asylum is a rectangular parallelepiped. Let us introduce coordinate system in such a way that two opposite corners of the asylum have coordinates $(0, 0, 0)$ and (W, H, D). Then position of the i-th ghost can be described in this system as sphere centered at point with coordinates (X_i, Y_i, Z_i) and having radius R_i. Second line of the input file contains integer number N — number of ghosts in the asylum ($0 \leqslant N \leqslant 10$). i-th of the next N lines contains four numbers X_i, Y_i, Z_i and R_i ($-1000 < X_i, Y_i, Z_i < 2000$, $0 \leqslant R_i \leqslant 1000$). Though new ghost cannot intersect other ghosts and borders of the asylum, old ghosts may violate these rules and cross each other and boundaries of the asylum. However, it is guaranteed that at least some part of a ghost is contained inside the asylum. Ghost of radius 0 is supposed to consist of one point and cannot lie inside your ghost.

Output

To the output file write three numbers — coordinates of a point where ghost of required maximal radius, centered at this point, may stay. If there are several such points, output coordinates of any of them (but only one). Your output will be considered correct if maximal radius of a ghost centered at the point found by your program will be within 1 mm of the optimal one. To avoid any precision problems we recommend to output coordinates with maximal possible accuracy. It is guaranteed that at least one solution with positive radius will exist.

Example

standard input	standard output

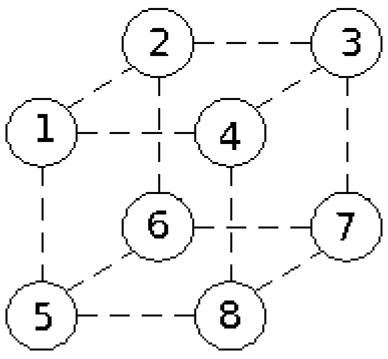
`9 13 15`
`2`
`3 9 6 1`
`4 8 3 5`
	`5 4 11`

In the example, the first ghost is inside the second one. Ghost of maximum radius of 4 can stay at the point (5, 4, 11) — in this case it touches three borders of the asylum and the second ghost.

Problem G. The Death Cube

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

A long time ago, in a Galaxy Far Far Again…



New dangerous mission undertake Jedi Knights. Latest and most powerful weapon of the Empire — the Death Cube — decided they to destroy. Not easy the mission is — new modified battle droids withstand them to, and very complex and awkward structure the Death Cube has. Specifically, many tiers consists the Cube of, and cellular structure each tier has. It is unimportant which face of the cube start you to mark out tiers from — every time distinguish tiers will you be able to. X tiers in one dimension of the Cube could count you, Y tiers in the second dimension, and Z tiers in the third one. Thus, not cube the Death Cube is, but rectangular parallelepiped instead, and XYZ cabins has the Cube. In other words, in a 3-D space can be allocated the Cube so that in integer point the center of every cabin is, and every integer point inside the Cube is the center of a cabin. In this system between any two cabins passageway exists if and only if exactly one unit the distance between the cabins is. For you convenience, to you example of the Cube with $X = Y = Z = 2$ is given.

The Cube to destroy, plan to place a bomb in every passageway of the Cube Jedi Knights. So to do, should in some cabin close to the outer surface of the Cube start the mission detachment of Jedis to. From cabin to cabin by moving, should mine they passageways passed, and in one of the cabins close to the outer surface of the Cube should finish their mission Jedi Knights. During their trip cannot separate from each other Jedis — only all together can overpower battle droids they. That what your job is — find the shortest way which should overpass Jedis to.

Input

Several descriptions of Death Cubes input file may contain (since indefatigable the Emperor is, and construct new Cubes continues he to). Only three numbers — X, Y and Z — contains every line of input ($1 \leqslant X, Y, Z \leqslant 1000$).

Output

For every Cube, only length of the shortest way should output you, and the way itself will help to Jedis the Power find to. Output details in example will find you.

Example

standard input	standard output

`2 1 2`
`2 2 2`
	`Case #1: 4`
	`Case #2: 15`

The following way one of the possible solutions to the Cube from the second example (and from the picture) is: 1—2—3—4—1—5—6—2—3—7—6—5—8—4—8—7.

Problem H. Funny Game

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

Goryinyich and vot decided to play the following funny game.

Initially some natural number is written on the board. Players move in turns, Goryinyich starts the game. In each turn, player wipes the board out and writes new number equal to the difference between erased number and number chosen by the player. Player may choose any number between one and sum of squares of digits of number that is written on the board. If minimal number is written on the board, the player who got the number in his turn is considered to win. Minimal number is the smallest possible number with a given length and given sum of squares of its digits.

Consider the example. Suppose that number 20 is written on the board. Goryinyich may subtract any number from 1 to $2^2 + 0^2 = 4$ from it. Any of these possibilities makes the number minimal in the abovementioned sense, so Goryinyich wins the game. But after the game was proposed by Goryinyich, vot claimed that Goryinyich went crazy, oversolved problems from Ural championships and vot will never play a game with such cumbersome and tedious rules. Instead, he proposed another game which, in his opinion, is much simpler and more cheerful.

Goryinyich and vot agree on some natural number N. Then vot writes some integer number a from 1 to N on a piece of paper and Goryinyich, unaware of the number written by vot, writes number b (also from 1 to N). If $|a - b| = 1$, then vot pays \$1 to Goryinyich, otherwise Goryinyich pays the same to vot.

Your task is to find expected gain of Goryinyich. Of course, you should realize that the players are not stupid (actually, they even also participate in computer programming contests from time to time), so everybody will do his best to maximize expected gain from the game.

Input

Input file contains up to 10 lines, each containing number N from the problem statement ($1 \leqslant N \leqslant 50$).

Output

For every N in the input file write to the output file on a separate line the required number as an irreducible fraction. Minus sign, if necessary, should be in the numerator. Follow the format shown in the example below.

Example

standard input	standard output

`1`
`2`
	`Case #1: -1/1`
	`Case #2: 0/1`

Problem I. Sokoban

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 megabytes

Sokoban (warehouse keeper) is a transport puzzle in which the player pushes boxes around a maze, viewed from above, and tries to put them in designated locations. Only one box may be pushed at a time, and boxes cannot be pulled. The problem of solving Sokoban puzzles has been proven to be NP-hard.

From Wikipedia, the free encyclopedia

Probably, everybody at least once in his life played this famous game. And despite the fact the problem is NP-hard you are required to write a program which solves this puzzle for quite huge mazes; moreover, it is needed to find the best solution to the puzzle. Specifically, from all possible solutions you are to select the one with the smallest number of box pushes. From all such solutions you need the one that minimizes the total number of moves. To simplify your task a bit, only puzzles with one box will be given to your program to solve.

Input

Maze for solving is given in the input file, maze has a size up to 100×100 cells. Every line of input file represents one row of the maze, all lines have the same length (and equal to the width of the maze). Space character represents a passable cell, where box and sokoban may stay, and «#» character represents an impassable wall. «@» character represents sokoban itself, «$» represents a cell in which the box is initially located, and «.» represents a cell where sokoban should put the box. Cells in which sokoban, box and destination cell are initially located are passable. Input file will contain every of the characters «@», «$» and «.» exactly once. Maze will be closed in a sense that sokoban will be unable to leave it.

Output

If there is no solution for the given maze, write to the output file the only message «`Impossible.`». Otherwise write one of the best (in the abovementioned sense) solutions. Each move is represented by one of the four characters: «`u`» means moving up, «`r`» — right, «`d`» — down and «`l`» — left. Use capital letters (i.e. «`U`», «`R`», «`D`» and «`L`») to represent moves when sokoban pushes the box.

Examples

standard input	standard output

`####`
`#  ##`
`#@$ #`
`#.# #`
`#   #`
`#####`
	`ddrruuLulD`

`####`
`#  ##`
`#@$ #`
`#.# #`
`# # #`
`#####`
	`Impossible.`

Problem J. Droid formation

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

A long time ago (but somehow in the future), in a Galaxy Far Far Away…

— Your majesty, Jedi detachment almost finished to mine our new Death Cube! New battle droids are unable to restrain them! What to do!?

— Rest assured! What is the strength of every troop of droids?

— 12 droids, your majesty!

— Fools! I told you that a troop should have 4 variants of evolution but troop of 12 droids has 6! This perverts threads of the Power — and infeeds Jedis! Regroup the army — and Jedis will lose!

— Yes sir!

Number of variants of evolution of a troop of droids is the number of ways to draw it up in rows so that every row has the same number of droids. For example, a troop of 12 droids can be arranged in 1 row of 12 droids, 2 rows of 6 droids, 3 rows of 4 droids, 4 rows of 3 droids, 6 rows of 2 droids and 12 rows consisting of 1 droid each. So, as the Emperor noticed, there are 6 variants of evolution for this troop of droids.

You problem is more general — given the number K of favorable variants of evolution, find the smallest positive size of a troop of droids N which has this very number of variants of evolution.

Input

Input file contains only number K from the problem statement ($1 \leqslant K \leqslant 10^5$).

Output

Write to the output file the required number N. If there is no such number, write to the output file number 0 instead.

Example

standard input	standard output
4	6