Problem E: The Bookcase

No wonder the old bookcase caved under the massive piles of books Tom had stacked on it. He had better build a new one, this time large enough to hold all of his books. Tom finds it practical to have the books close at hand when he works at his desk. Therefore, he is imagining a compact solution with the bookcase standing on the back of the desk. Obviously, this would put some restrictions on the size of the bookcase, it should preferably be as small as possible. In addition, Tom would like the bookcase to have exactly three shelves for aesthetical reasons.

Wondering how small his bookcase could be, he models the problem as follows. He measures the height h_i and thickness t_i of each book i and he seeks a partition of the books in three non-empty sets S_1, S_2, S_3 such that $\left(\sum_{j=1}^{3} \max_{i \in S_j} h_i\right) \times \left(\max_{j=1}^{3} \sum_{i \in S_j} t_i\right)$ is minimized, i.e. the area of the bookcase as seen when standing in front of it (the depth needed is obviously the largest width of all his books, regardless of the partition). Note that this formula does not give the exact area of the bookcase, since the actual shelves cause a small additional height, and the sides cause a small additional width. For simplicity, we will ignore this small discrepancy.

Thinking a moment on the problem, Tom realizes he will need a computer program to do the job.

Input

The input begins with a positive number on a line of its own telling the number of test cases (at most 20). For each test case there is one line containing a single positive integer N, $3 \leq N \leq 70$ giving the number of books. Then N lines follow each containing two positive integers h_i, t_i, satisfying $150 \leq h_i \leq 300$ and $5 \leq t_i \leq 30$, the height and thickness of book i respectively, in millimeters.

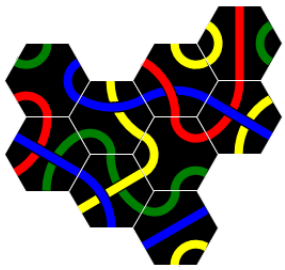
Output

For each test case, output one line containing the minimum area (height times width) of a three-shelf bookcase capable of holding all the books, expressed in square millimeters.

Sample input	**Sample output**
2 4 220 29 195 20 200 9 180 30 6 256 20 255 30 254 15 253 20 252 15 251 9	18000 29796

Problem G: Tantrix

Tantrix is a two player game played with 56 hexagonal tiles. Each tile contains three *links* in different colours. Both players have five tiles in hand and take turns in placing them on the playing field. The figure to the right shows how the game could have progressed after nine played tiles.

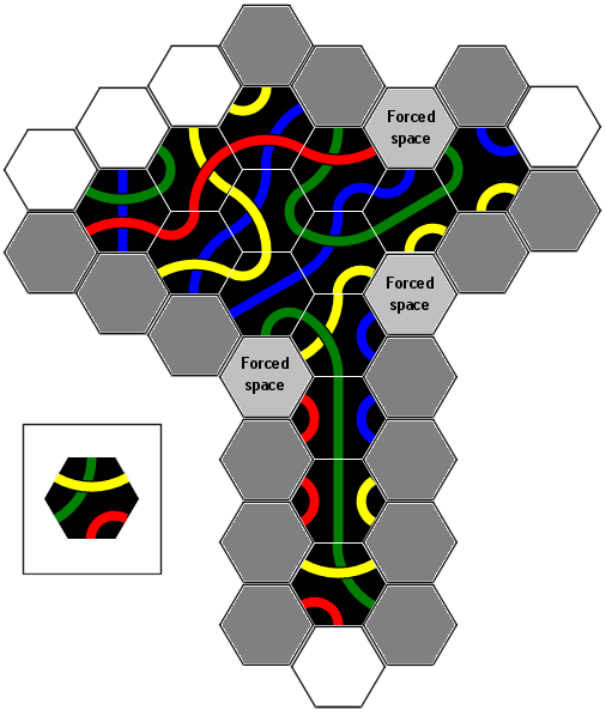


There are four different link colours: red, green, yellow and blue. No two tiles are identical, and no tile is rotation symmetric. A tile will be described in the input as a six letter string, specifying the link colours in clockwise direction. The uppercase letters 'R', 'G', 'Y' and 'B' will be used for red, green, yellow and blue, respectively.

In this problem, a move is defined as placing one of the tiles in hand somewhere on the playing field, subject to these rules:

1. A tile must always be placed next to tiles already played.
2. The links in all touching tiles must match colour.
3. An empty space which is surrounded by three tiles is called a *forced space*. If the player can place one of his tiles in a forced space, he must do so. If there are several forced spaces, and several ways to place a tile in a forced space, he may select any of those.
4. It's not allowed to place a tile so that a forced space is created containing three links of the same colour (since no tile could ever be placed there).
5. The two sides along a forced space are called *controlled sides*. It's not allowed to place a tile along a controlled side.

If there are one or more forced spaces and the player can't place any of his tiles in hand in those spaces, he will have to play any other legal move. Note that a player may not be allowed to place a tile in a forced space due to rule 4.



The figure on the right illustrates these rules. There are three forced spaces. The interposed tile may not be placed in the lower left forced space, as that would create a new forced space with three red links. The dark gray spaces lie on controlled sides created by the forced spaces; no tiles may be placed there. If the player to move can't place a tile in any of the three forced spaces, he must place a tile in any of the white spaces.

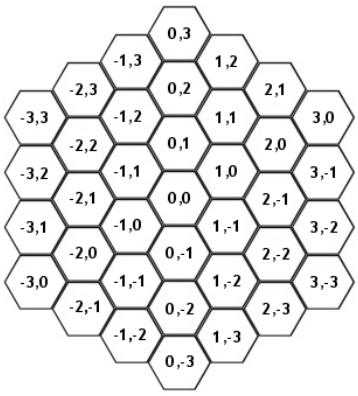
Your task is to count the number of legal moves the player to move has,

given the position and orientation of already played tiles and the tiles in hand for the player to move. If a tile can be placed at several locations, or in several orientations, each such combination is counted as a distinct move.

Input

The first line in the input will contain the number of cases (at most 50).

Each case begins with a single line containing an integer n ($1 \leq n \leq 20$), the number of tiles that have already been played. Then follow n lines containing the coordinates and description of these tiles. The first character in the tile description belongs to the link facing up; the remaining colours follow as per usual in clockwise direction. Then follows a line with the description of the five tiles in hand, the tile descriptions being separated with a single space.



The mapping between the spaces and the coordinates is shown in the figure below (note that the playing field is infinite and not restricted to these coordinates). All tiles in the input will be valid and distinct. The layout will represent a position that could have arisen from a legal game. One of the played tiles will have coordinates $0, 0$.

Output

For each test case, output a single line containing an integer: the number of legal moves.

Sample input	**Sample output**
2 6 0 0 BRYRBY 1 0 GRGBRB -1 1 GGYBYB 0 1 YYBBGG -2 2 YYBGBG -3 3 BYGYGB BBRRGG GBYBYG RBRBGG GYBGBY GRBBRG 4 0 0 BYYGBG -1 1 GRGBBR 1 0 YRBRYB 2 0 YGGRRY RBBRYY GBGYBY YBBRYR YBYBRR RBBRGG	46 2

2591 - The Tree Movers

North America - North Central - 2002/2003

Given two binary search trees, A and B, with nodes identified by (that is, having keys equal to) positive, non-zero integers, and the use of commands ``delete *K* '' and ``add *K* '' (defined below), what is the smallest number of commands that can be used to transform tree A into tree B?

Recall that in a binary search tree, the keys of all nodes in the left subtree of a node with key *K* must be less than *K* . Similarly, the keys of all nodes in the right subtree of a node with key *K* must be greater than *K* . There are no duplicate nodes.

The ``delete *K* '' command will delete the tree (or subtree) with its root at the node with the key *K* . Deleting the root of the entire tree leaves an empty tree. The ``add *K* '' command will add a new node identified by the integer *K* . This node will naturally be a leaf node.

Since we seek to transform tree A into tree B, it follows that commands will be applied only to tree A; tree B is ``read only".

It is easy to see that it should never require more than $N + 1$ commands to achieve the transformation of A into B, since deletion of the root node of tree A followed by the addition of one node for each of the N nodes in B (in the proper order) will achieve the desired goal. Equally easy to determine is the minimum number of commands required: if A and B are identical, then zero commands are required.

Input

There will be multiple input cases. For each case, the input contains the description of tree A followed by the description of tree B. Each tree description consists of an integer N that specifies the number of nodes in the tree, following by the keys of the N nodes in an order such that N ``add" commands would create the tree. The last case is followed by the integer `-1'. No node will have a key larger than 10^9 , and N will be no larger than 100.

Output

For each case, display a single line containing the input case number (1, 2,...) and the number of commands required to transform tree A into tree B, formatted as shown in the examples below.

Sample Input

```
4 5 2 7 4 6 5 3 7 1 4 9
0 0
1 100 0
0 1 100
3 100 49 37 2 200 152
-1
```

Sample Output

```
Case 1: 5 commands.
Case 2: 0 commands.
Case 3: 1 command.
```

```
Case 4: 1 command.
Case 5: 3 commands.
```

North Central 2002-2003

The Tower of Babylon

Perhaps you have heard of the legend of the Tower of Babylon. Nowadays many details of this tale have been forgotten. So now, in line with the educational nature of this contest, we will tell you the whole story:

The babylonians had *n* types of blocks, and an unlimited supply of blocks of each type. Each type−*i* block was a rectangular solid with linear dimensions (x_i, y_i, z_i). A block could be reoriented so that any two of its three dimensions determined the dimensions of the base and the other dimension was the height. They wanted to construct the tallest tower possible by stacking blocks. The problem was that, in building a tower, one block could only be placed on top of another block as long as the two base dimensions of the upper block were both strictly smaller than the corresponding base dimensions of the lower block. This meant, for example, that blocks oriented to have equal−sized bases couldn't be stacked.

Your job is to write a program that determines the height of the tallest tower the babylonians can build with a given set of blocks.

Input and Output

The input file will contain one or more test cases. The first line of each test case contains an integer *n*, representing the number of different blocks in the following data set. The maximum value for *n* is 30. Each of the next *n* lines contains three integers representing the values x_i, y_i and z_i.

Input is terminated by a value of zero (0) for *n*.

For each test case, print one line containing the case number (they are numbered sequentially starting from 1) and the height of the tallest possible tower in the format "`Case` *case*`: maximum height =` *height*"

Sample Input

```
1
10 20 30
2
6 8 10
5 5 5
7
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
5
31 41 59
26 53 58
97 93 23
84 62 64
33 83 27
0
```

Sample Output

```
Case 1: maximum height = 40
Case 2: maximum height = 21
Case 3: maximum height = 28
Case 4: maximum height = 342
```

3385 - Leaping Lizards

North America - Pacific Northwest - 2005/2006

Your platoon of wandering lizards has entered a strange room in the labyrinth you are exploring. As you are looking around for hidden treasures, one of the rookies steps on an innocent-looking stone and the room's floor suddenly disappears! Each lizard in your platoon is left standing on a fragile looking pillar, and a fire begins to rage below...

Leave no lizard behind! Get as many lizards as possible out of the room, and report the number of casualties.

The pillars in the room are aligned as a grid, with each pillar one unit away from the pillars to its east, west, north and south. Pillars at the edge of the grid are one unit away from the edge of the room (safety). Not all pillars necessarily have a lizard. A lizard is able to leap onto any unoccupied pillar that is within d units of his current one. A lizard standing on a pillar within leaping distance of the edge of the room may always leap to safety... but there's a catch: each pillar becomes weakened after each jump, and will soon collapse and no longer be usable by other lizards. Leaping onto a pillar does not cause it to weaken or collapse; only leaping off of it causes it to weaken and eventually collapse. Only one lizard may be on a pillar at any given time.

Input

The input file will begin with a line containing a single integer representing the number of test cases, which is at most 25. Each test case will begin with a line containing a single positive integer n representing the number of rows in the map, followed by a single non-negative integer d representing the maximum leaping distance for the lizards. Two maps will follow, each as a map of characters with one row per line. The first map will contain a digit (0-3) in each position representing the number of jumps the pillar in that position will sustain before collapsing (0 means there is no pillar there). The second map will follow, with an `L' for every position where a lizard is on the pillar and a `.' for every empty pillar. There will never be a lizard on a position where there is no pillar.

- Each input map is guaranteed to be a rectangle of size $n * m$, where $1 \leq n \leq 20$ and $1 \leq m \leq 20$.
- Leaping distance is guaranteed to be in the range [1, 3].

Output

For each input case, you should return the number of lizards that could not escape. There should be a newline after each case, and your output format should follow the sample provided below.

Sample Input

```
4
3 1
1111
1111
1111
LLLL
LLLL
LLLL
3 2
00000
```

```
01110
00000
.....
.LLL.
.....
3 1
00000
01110
00000
.....
.LLL.
.....
5 2
00000000
02000000
00321100
02000000
00000000
........
........
..LLLL..
........
........
```

Sample Output

```
Case #1: 2 lizards were left behind.
Case #2: no lizard was left behind.
Case #3: 3 lizards were left behind.
Case #4: 1 lizard was left behind.
```

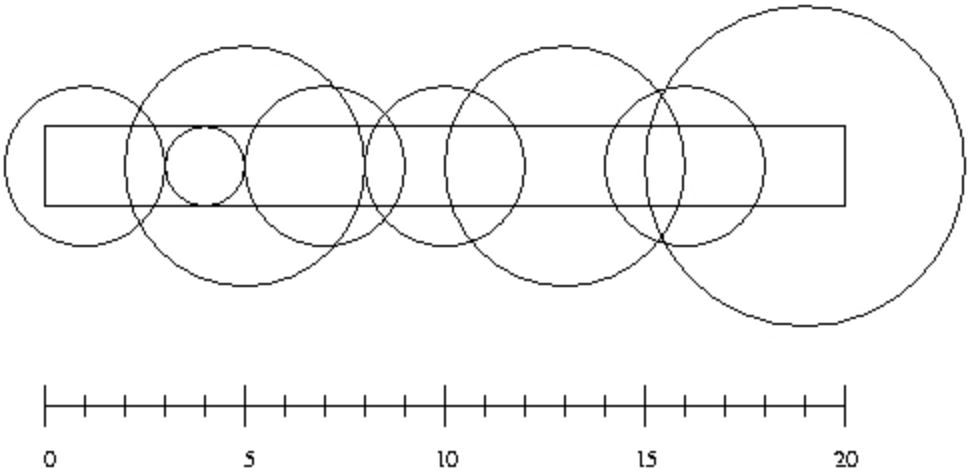
Pacific Northwest 2005-2006

Problem E
Watering Grass

Input: standard input
Output: standard output
Time Limit: 3 seconds

n sprinklers are installed in a horizontal strip of grass *l* meters long and *w* meters wide. Each sprinkler is installed at the horizontal center line of the strip. For each sprinkler we are given its position as the distance from the left end of the center line and its radius of operation.

What is the minimum number of sprinklers to turn on in order to water the entire strip of grass?



Input

Input consists of a number of cases. The first line for each case contains integer numbers *n*, *l* and *w* with *n* <= 10000. The next *n* lines contain two integers giving the position of a sprinkler and its radius of operation. (The picture above illustrates the first case from the sample input.)

Output

For each test case output the minimum number of sprinklers needed to water the entire strip of grass. If it is impossible to water the entire strip output -1.

Sample input

8 20 2

```
5 3
4 1
1 2
7 2
10 2
13 3
16 2
19 4
3 10 1
3 5
9 3
6 1
3 10 1
5 3
1 1
9 1
```

Sample Output

```
6
2
-1
```

(Regionals 2002 Warm-up Contest, Problem setter: Piotr Rudnicku)

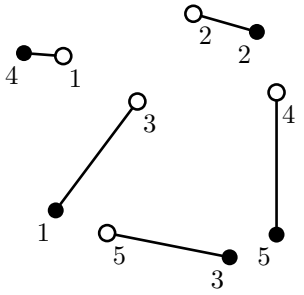
Problem A. Ants

Input file: `ants.in`
Output file: `ants.out`

Young naturalist Bill studies ants in school. His ants feed on plant-louses that live on apple trees. Each ant colony needs its own apple tree to feed itself.

Bill has a map with coordinates of n ant colonies and n apple trees. He knows that ants travel from their colony to their feeding places and back using chemically tagged routes. The routes cannot intersect each other or ants will get confused and get to the wrong colony or tree, thus spurring a war between colonies.

Bill would like to connect each ant colony to a single apple tree so that all n routes are non-intersecting straight lines. In this problem such connection is always possible. Your task is to write a program that finds such connection.



On this picture ant colonies are denoted by empty circles and apple trees are denoted by filled circles. One possible connection is denoted by lines.

Input

The first line of the input file contains a single integer number n ($1 \le n \le 100$) — the number of ant colonies and apple trees. It is followed by n lines describing n ant colonies, followed by n lines describing n apple trees. Each ant colony and apple tree is described by a pair of integer coordinates x and y ($-10\,000 \le x, y \le 10\,000$) on a Cartesian plane. All ant colonies and apple trees occupy distinct points on a plane. No three points are on the same line.

Output

Write to the output file n lines with one integer number on each line. The number written on i-th line denotes the number (from 1 to n) of the apple tree that is connected to the i-th ant colony.

Sample input and output

ants.in	ants.out
5 -42 58 44 86 7 28 99 34 -13 -59 -47 -44 86 74 68 -75 -68 60 99 -60	4 2 1 5 3

Problem E. Equation

Input file: `equation.in`
Output file: `equation.out`

Your task is to solve an equation of the form $f(x) = 0$ where $f(x)$ is written in postfix notation with numbers, operations `+`, `-`, `*`, `/`, and at most one occurrence of a variable x.

For example, $f(x)$ for an equation $(4x + 2)/2 = 0$ is written as:

```
4 X * 2 + 2 /
```

The solution for $f(x) = 0$ is $x = -1/2$.

Input

The input file consists of a single line with at most 30 tokens separated by spaces. Each token is either:

- a digit from `0` to `9`;
- an operation `+`, `-`, `*`, or `/`;
- an uppercase letter `X` that denotes variable x.

The input file contains a correct representation of $f(x)$ in postfix notation where token `X` occurs at most once. There is no division by a constant zero in this equation, that is, there always exists a value of x, such that $f(x)$ can be evaluated without division by zero.

Output

Write to the output file:

- `X =` p/q if equation $f(x) = 0$ has a single solution that can be represented with a simple fraction p/q, where p and q are coprime integer numbers and q is positive.
- `NONE` if equation $f(x) = 0$ has no solution;
- `MULTIPLE` if equation $f(x) = 0$ has multiple solutions.

Sample input and output

equation.in	equation.out
4 X * 2 + 2 /	X = -1/2
2 2 *	NONE
0 2 X / *	MULTIPLE

Problem C. Cellular Automaton

Input file: `cell.in`
Output file: `cell.out`

A *cellular automaton* is a collection of cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules that describe the new state of a cell based on the states of neighboring cells. The *order of the cellular automaton* is the number of cells it contains. Cells of the automaton of order n are numbered from 1 to n.

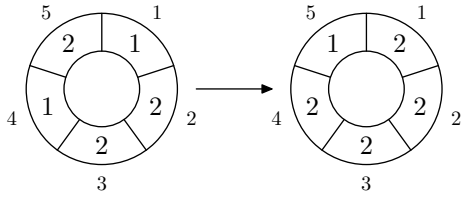
The *order of the cell* is the number of different values it may contain. Usually, values of a cell of order m are considered to be integer numbers from 0 to $m - 1$.

One of the most fundamental properties of a cellular automaton is the type of grid on which it is computed. In this problem we examine the special kind of cellular automaton — circular cellular automaton of order n with cells of order m. We will denote such kind of cellular automaton as *n,m-automaton*.

A distance between cells i and j in n,m-automaton is defined as $\min(|i-j|, n-|i-j|)$. A *d-environment of a cell* is the set of cells at a distance not greater than d.

On each *d-step* values of all cells are simultaneously replaced by new values. The new value of cell i after d-step is computed as a sum of values of cells belonging to the d-enviroment of the cell i modulo m.

The following picture shows 1-step of the 5,3-automaton.



The problem is to calculate the state of the n,m-automaton after k d-steps.

Input

The first line of the input file contains four integer numbers n, m, d, and k ($1 \le n \le 500$, $1 \le m \le 1\,000\,000$, $0 \le d < \frac{n}{2}$, $1 \le k \le 10\,000\,000$). The second line contains n integer numbers from 0 to $m - 1$ — initial values of the automaton's cells.

Output

Output the values of the n,m-automaton's cells after k d-steps.

Sample input and output

cell.in	cell.out
5 3 1 1 1 2 2 1 2	2 2 2 2 1
5 3 1 10 1 2 2 1 2	2 0 0 2 2