1. Aligning Text

Introduction

You have been assigned the critical mission of transmitting messages to a recently discovered alien race. However, the messages you were given to send have not been properly aligned for the aliens' computer screens, and as a result are rendered unreadable.

Task

You need to write a program to fix the messages before they are transmitted. Given a message consisting of N lines, your program should print each line such that it is either left- or right-aligned for a certain screen width W.

Example

Suppose you are given a message containing the line `hello`. If your task is to print it left-aligned then you should print the string `hello`, followed by a newline character. If it needs to be right-aligned for a width of ten, then you should print five leading spaces and then the string `hello` (so that the total number of characters printed is equal to the screen's width), followed by a newline character.

Input

The first line of input contains an integer T, the number of test cases. T test cases follow.

The first line of each test case contains two integers, N and W (separated by a space), followed by the type of text alignment required (`LEFT` or `RIGHT`). The next N lines contain arbitrary text. No line will contain more than W characters (excluding the terminating newline character), nor will any line contain leading or trailing whitespace.

Sample input

```
2
1 10 LEFT
hello
1 10 RIGHT
hello
```

Output

For each test case, output a line of the form

```
Case #X:
```

where X is the test case number, starting from 1, followed by N lines. If the alignment type is `LEFT`, then each line of the input message should be printed with no leading or trailing whitespace. If the alignment type is `RIGHT`, then each line should have exactly enough leading whitespace to make the total number of characters in the line equal to the screen width W.

Sample output

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

Constraints

- $1 \leq T \leq 20$
- $1 \leq N \leq 10$
- $1 \leq W \leq 80$

2. Divisibility Tricks

Introduction

Your computer science lecturer has discovered that many computer science students do not know the so-called "divisibility tricks" which allow one to quickly determine (without dividing) whether a number is divisible by a small factor. For example,

- If the sum of a number's digits is divisible by three, so is the original number.
- If the last two digits of a number form a number divisible by four, then the original number is also divisible by four.

So, 189 084 is divisible by three because $1+8+9+0+8+4 = 30$ is; it is also divisible by four since 84 is.

To practice one of these rules, your lecturer has given you a game where you must find numbers that are divisible by twelve. This game is not much fun, so you have decided to write a program to play it for you.

Task

Given the output a large number, D, your task is to find the largest prefix, D_N, which is divisible by 12. The prefix D_N is defined as a number which consists of the initial N digits of D in order. Your friend has prepared T test cases to ensure that your program is correct. For the purposes of these tests, it is not neccessary to output the entire prefix, D_N. Instead, only the length, N, of the longest prefix divisible by twelve is required.

Example

Suppose you are given the number 1236 as input. We can see that the prefixes 12 and 1236 are divisible by 12, but 1236 is the largest so we output 4. Now suppose we are given the number 1 234 567 890. Only the prefixes 12 and 123 456 are divisible by 12. In this case the answer is 6.

Input

The first line of input contains the integer T, the number of test cases.

The following T lines each contain a single number D.

Sample input

```
2
5604
1234567890
```

Output

For each test case, output a line of the form

```
Case #X: N
```

where X is the test case number, starting from 1 and N is the length of the largest prefix. If no prefix is divisible by 12, output 0.

Sample output

```
Case #1: 4
Case #2: 6
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 20$
- $1 \leq$ length of each $D \leq 100\,000$
- the first digit of D is not 0

3. Marching Band

Introduction

Your local school is organising a marching band to celebrate its anniversary. Unfortunately, the parade is very complex and some of the marching band members are finding it difficult to memorise the routine. You have been tasked with writing a program to simulate the band's movements throughout the parade to help them practice.

Task

The marching band is organised as rectangular grid of musicians. The grid has R rows and C columns. Each musician has been given a unique identifier. The musicians in the first row have the values 1 to C in that order, the second row has the values $C + 1$ to $2C$ in that order, and so on. This is shown for the 3×3 case in the diagram below.

$$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$$

Your task is to perform a list of interspersed operations and queries on the arrangement. An operation will alter the arrangement of musicians in the grid, while a query will ask you for some information about the arrangement. There are five different operations that may be performed: swapping two rows, swapping two columns, reversing the order of each row, reversing the order of each column, and transposing the arrangement.

Transposition means that each musician standing at row r and column c will move immediately to row c and column r. This could change the dimensions of the grid! An example of transposition for a 2×3 grid is given below.

$$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix} \Rightarrow \begin{pmatrix}1 & 4\\2 & 5\\3 & 6\end{pmatrix}$$

There are two types of queries that may be asked: which musician is at a given position, and what is the current position for a given musician. Note that the identifiers in the grid are unique, so these queries have unique answers. Rows are indexed from 1 to R and columns are indexed from 1 to C.

Your school has not yet decided on which procession to use and has given you T test cases which your program must correctly answer. Each test case uses a new grid of possibly different dimensions and a new set of operations and queries that must be performed.

Example

Suppose you are given a 3×3 grid. The initial values in the arrangement are:

$$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$$

We are now asked to determine the musician at row 1 and column 2: it is musician 2. If we are asked to swap row 1 and row 2, the arrangement becomes:

$$\begin{pmatrix}4 & 5 & 6\\1 & 2 & 3\\7 & 8 & 9\end{pmatrix}$$

Next we are asked to determine the position of musician 2 in the grid. The previous swap has moved the musician to their new location at row 2 and column 2. Now suppose we are asked to perform a transposition, the resulting arrangement is:

$$\begin{pmatrix}4 & 1 & 7\\5 & 2 & 8\\6 & 3 & 9\end{pmatrix}$$

Finally, we are asked to determine the musician at row 1 and column 2 of the new grid: it is musician 1.

Input

The first line of input contains the integer T, the number of test cases. This is followed by T test cases.

The first line of each test case contains three integers: R, C and N. R and C are the number of rows and columns in the initial grid. N is total number of operations and queries that must be performed. Each of the following N lines describe one operation or query that is to be run. These lines will look like one of the following:

```
r r1 r2
```

where $r1$ and $r2$ are two integers from 1 to R. This operation indicates that rows $r1$ and $r2$ should be swapped. For example, swapping rows 1 and 2 on the starting 3×3 grid yields:

$$\begin{pmatrix}4 & 5 & 6\\1 & 2 & 3\\7 & 8 & 9\end{pmatrix}$$

```
c c1 c2
```

where $c1$ and $c2$ are two integers from 1 to C. This operation indicates that columns $c1$ and $c2$ should be swapped. Swapping columns 1 and 2 on a starting 3×3 grid results in the following:

$$\begin{pmatrix} 2 & 1 & 3 \\ 5 & 4 & 6 \\ 8 & 7 & 9 \end{pmatrix}$$

```
R
```

This operation indicates that the elements in **every** row should be reversed. After a row reversal the original grid becomes:

$$\begin{pmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{pmatrix}$$

```
C
```

This operation indicates that the elements in **every** column should be reversed. After a column reversal the original grid becomes:

$$\begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$$

```
T
```

This operation indicates that the grid should be transposed. After a transposal the original grid becomes:

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

```
V r c
```

This query indicates the value v at row r and column c should be output.

```
P v
```

This query indicates that the row r and column c that contain the value v should be output.

Sample input

```
1
3 3 5
V 1 2
r 1 2
P 2
T
V 1 2
```

Output

At the start of each test case, output a line of the form

```
Case #X:
```

where X is the test case number, starting from 1. Each subsequent line will contain the output of a subsequent query from the test case. Do not output anything for an operation.

Sample output

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

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 20$
- $1 \leq R \leq 10\,000$
- $1 \leq C \leq 10\,000$
- $2 \leq N \leq 2\,000$

4. Satellite Navigation

Introduction

A local manufacturer of navigation (GPS) systems has almost completed the software for a car navigation system. The only step remaining is to integrate the voice synthesis module (for reading directions) with the path-finding module.

Unfortunately, a computer virus infected the computers of the team working on the path-finding module, and all copies of its source code have been destroyed. All that remains is a compiled version that produces some debugging output: an ASCII-art map with the route marked.

You need to convert this debugging output to text for input into the voice synthesis system.

Task

Write a program that will take a map with a marked route and produce textual directions in English.

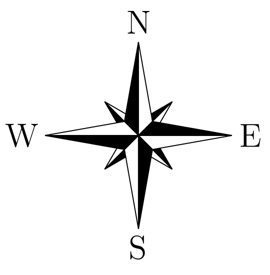
Input

The first line of the input contains an integer T, the number of test cases. Each test case consists of a line containing an integer L, followed by L lines of map data with a marked route.

The map data is a grid with characters marking roads, with different characters for sections of road which are used by the route. Each character represents a 50 m by 50 m square. The width of the roads within the squares are negligible. The characters are as follows:

	off-route	on-route
north-south road	:	|
west-east road	.	-
intersection or corner	+	#

The start point of the route will be marked by a tilde (`~`) if is on a west-east road and by `I` if it is on a north-south road. The end point will not be marked differently from the rest of the route. The start and end points will not be on intersections or corners. The start and end points are in the centre of their respective squares.



Sample input

```
2
8
~-#....+
  |    :
  |    :
  |    :
..#----#----#...
     | :    |
     |      |
     #------#
4
I    :         :
#----#------#..+
        :   |
        +---#
```

Output

For each test case, first output

```
Case #X:
```

where X is the test case number, starting from 1. Then, on a new line, output

```
Head H
```

where H is `north`, `east`, `south` or `west`.

For each turn on the route, output on a new line

```
Take the N D
```

where D is `left` or `right` and N is the number of the turn as a word (e.g. if this is the second place where it is possible to turn left, output `Take the second left`). However, if the turn is at the end of the road, instead output

```
Turn D at the end of the road
```

After the last turn, output

```
Continue for M metres
```

where M is the distance from the last turn (or the start point if there were no turns) to the end point. However, if the end point is at the end of the road, instead output

```
Continue to the end of the road
```

Finally, output

```
Stop
```

followed by a blank line

4. Satellite Navigation

Sample output

```
Case #1:
Head east
Take the first right
Turn left at the end of the road
Take the second right
Turn right at the end of the road
Turn right at the end of the road
Continue to the end of the road
Stop

Case #2:
Head south
Turn left at the end of the road
Take the first right
Turn right at the end of the road
Continue for 150 metres
Stop
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \le T \le 20$
- $1 \le L \le 100$
- $1 \le$ width of each line ≤ 100
- no `+` or `#` is adjacent (either west-east or north-south) to another `+` or `#`
- there is exactly one continuous route on each map
- there is only one possible route corresponding to the markings on the map (i.e. the route is unambiguous)

Language

For reference, the first hundred ordinal numbers are written as follows:

```
                        tenth
first                   eleventh
second                  twelfth
third                   thirteenth
fourth                  fourteenth
fifth                   fifteenth
sixth                   sixteenth
seventh                 seventeenth
eighth                  eighteenth
ninth                   nineteenth

twentieth               thirtieth
twenty-first            thirty-first
twenty-second           thirty-second
twenty-third            thirty-third
twenty-fourth           thirty-fourth
twenty-fifth            thirty-fifth
twenty-sixth            thirty-sixth
twenty-seventh          thirty-seventh
twenty-eighth           thirty-eighth
twenty-ninth            thirty-ninth
                        fiftieth
fortieth
forty-first             fifty-first
forty-second            fifty-second
forty-third             fifty-third
forty-fourth            fifty-fourth
forty-fifth             fifty-fifth
forty-sixth             fifty-sixth
forty-seventh           fifty-seventh
forty-eighth            fifty-eighth
forty-ninth             fifty-ninth

sixtieth                seventieth
sixty-first             seventy-first
sixty-second            seventy-second
sixty-third             seventy-third
sixty-fourth            seventy-fourth
sixty-fifth             seventy-fifth
sixty-sixth             seventy-sixth
sixty-seventh           seventy-seventh
sixty-eighth            seventy-eighth
sixty-ninth             seventy-ninth

eightieth               ninetieth
eighty-first            ninety-first
eighty-second           ninety-second
eighty-third            ninety-third
eighty-fourth           ninety-fourth
eighty-fifth            ninety-fifth
eighty-sixth            ninety-sixth
eighty-seventh          ninety-seventh
eighty-eighth           ninety-eighth
eighty-ninth            ninety-ninth
hundredth
```

5. Connecting Circuits

Introduction

You have been asked to consult on the circuit board design for a new electronic product. A circuit board design consists of a number of input and output points which need to be connected by wires. There are a number of possible designs and your help is needed to determine which one will be the most suitable.

Task

There are N input and N output points on the circuit board. The input points are labeled A_1 to A_N and are positioned along the top edge of the rectangular circuit board. The output points are labeled B_1 to B_N and are positioned along the bottom edge. The position of each input and output point is specified by an integer which indicates how far along the edge of the circuit board it is located (with 0 being the left-most edge of the board and 1 000 000 the right-most edge). **The order in which the input and output points are positioned is not related to the order in which they are labeled.** No two input points will have the same position on the top edge of the circuit board, and similarly for the output points.

The input and output points of the circuit board need to be connected, but each input point can only be connected to its corresponding output point labeled with the same number (i.e. the input point A_i can only be connected to the output point B_i.) These connections can only be made using a single wire that connects A_i to B_i in a straight line. To avoid any electrical interference, these connecting wires are not allowed to cross.

Given these conditions, it is usually not possible to connect all input points to their corresponding output points. Your task, given a number of circuit board designs, is to determine the maximum number of input/output point pairs that can connected on each separate board without any wires crossing.

Example

Suppose you are given a design with 5 input points (at positions 4, 5, 3, 1 and 6 respectively) and 5 output points (at positions 6, 3, 1, 2 and 5 respectively). You could connect points A_3 to B_3 and A_1 to B_1, as shown in Figure 1. No further connections can be made at this point without causing some wires to cross, which would give you a total of two connections made.

However, a better set of connections is possible: connect A_4 to B_4, A_2 to B_2 and A_5 to B_5 (this is shown in Figure 2). No further connections can be made, but this is a total of three connections, which is the greatest number

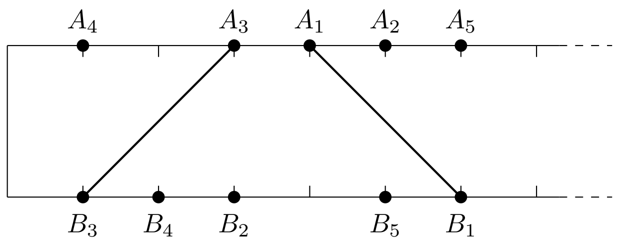


Figure 1

possible for this circuit board. Note that you could choose to connect A_3 to B_3 instead of A_4 to B_4 to achieve the same total number of connections. There may be multiple different ways to achieve the maximum number of connections for a given circuit board, but you are only required to output this maximum.

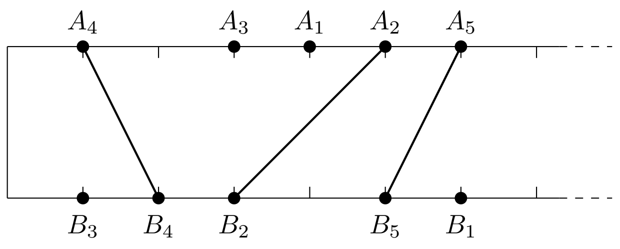


Figure 2

Input

The first line of input contains the integer T, the number of test cases. The following lines describe each test case in turn. Each test case consists of three lines.

The first line of a test case contains a single integer N, the number of input and output points on the circuit board given in this test case. The second line consists of N space-separated integers, a_1 to a_N, describing the positions of the input points on the top edge of the circuit board: a_i is the position of point A_i on the top edge. Similarly, the third line consists of N space-separated integers, b_1 to b_N, which describe the positions of the output points on the bottom edge of the circuit board: b_i is the position of point B_i on the bottom edge.

Sample input

```
2
5
4 5 3 1 6
6 3 1 2 5
4
1 2 3 4
4 3 2 1
```

Output

For each test case, output a line of the form

```
Case #X: C
```

where X is the test case number, starting from 1, and C is the maximum number of connections that can be made between pairs of input and output points.

Sample output

```
Case #1: 3
Case #2: 1
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 20$
- $1 \leq N \leq 200$
- $1 \leq a_i, b_i \leq 1\,000\,000$ for all i
- all a_i are unique, and all b_i are unique. ($a_i \neq a_j$ and $b_i \neq b_j$ for all $i \neq j$)