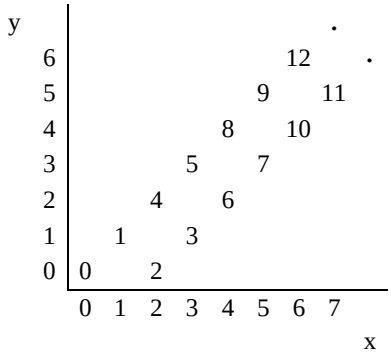
Problem A

(Program filename: A.CPP or A.PAS)

Number Steps

Starting from point (0,0) on a plane, we have written all non-negative integers 0,1,2, … as shown in the figure. For example, 1, 2, and 3 has been written at points (1,1), (2,0), and (3, 1) respectively and this pattern has continued.



You are to write a program that reads the coordinates of a point (x, y), and writes the number (if any) that has been written at that point. (x, y) coordinates in the input are in the range 0…5000.

Input (filename: A.IN)

The first line of the input is *N*, the number of test cases for this problem. In each of the *N* following lines, there is x, and y representing the coordinates (x, y) of a point.

Output (filename: A.OUT)

For each point in the input, write the number written at that point or write `No Number` if there is none.

Sample Input

```
3
4 2
6 6
3 4
```

Sample Output

```
6
12
No Number
```

Problem B

(Program filename: B.CPP or B.PAS)

Puzzlestan

His Excellency the ambassador of Puzzlestan to Tehran is giving a reception (party) on the occasion of Puzzlestan national day on the second of November. Ambassadors of all missions to Tehran are invited. At the entrance to reception hall each guest leaves his belongings (overcoat, hat, …, which we will call items afterwards) with the attendant who knows the host ambassador loves puzzle games. While waiting for the reception to end, the attendant prepares a program on his laptop that helps him find what belongs to whom. He asks each guest to give some statements that:

- Relates two items belonging to the same guest.
- Indicates two items that do not belong to the same guest.

Suppose there are N kinds of items and M guests, and assume each guest has exactly one item from each kind (i.e. each guest has one overcoat, one hat, …). Each of $N \times M$ items is given a unique name, which is a single letter (case-sensitive). We can then represent all items of kind i ($1 \le i \le N$) by a string of length M. We call this, ith group. For example, ith group can be ABCD. In this case, its first item is A, its second item is B, and so on.

Input (filename: B.IN)

The first line includes the number of test cases (at most 20). First line of each test case includes two positive integers; the first one being N ($1 \le N \le 7$), the number of item kinds, and the second is M ($1 \le M \le 7$), the number of guests. In each of the next N lines, there is one string of length M, representing one group (different items of the same kind). The rest of the input includes several lines of statements. Each statement is of form *i j X k r*, which means that the jth item of the ith group and the rth item of the kth group belong to the same guest (if X is R) or do not belong to the same guest (if X is N). The last line of each test case is a dummy statement 0 0 R 0 0.

Output (filename: B.OUT)

For each test case, the program should produce M lines, line i ($1 \le i \le$ M) starts with the letter indicating the ith member of the first group in the input test case, then the related member of the second set and so on until that of the last set. (So the letters in the first column are the same as the letters in the first group, preserving order). Output for consecutive test cases must be separated by exactly one blank line.

Sample Input

```
1
3 4
ABCD
EFGH
IJKL
1 1 R 3 2
1 2 N 2 2
2 2 R 3 4
1 3 R 2 3
1 1 N 2 4
3 1 R 1 3
0 0 R 0 0
```

Sample Output

```
AEJ
BHK
CGI
DFL
```

Problem C

(Program filename: C.CPP or C.PAS)

Dynamic Declaration Language (DDL)

DDL is a very simple programming language in which variables are dynamically declared at run time. All variables in DDL are of the signed integer type within the range –9999…9999. There are up to five types of statements in a DDL program (each statement is in a separate program line, and the first statement is in line 1):

1. `Dcl <id>`

   `Dcl` is a keyword specifying a declaration statement. `id` is a single (case-sensitive) letter designating a DDL variable. For example `Dcl x` when executed correctly, allocates memory for variable `x`, and sets its value to zero.

2. `<id> = <ic>`

   This is an assignment statement, where `id` is a DDL variable, and `ic` is a literal integer constant in the range (0…9999). For example `x = 2000` when executed correctly, changes value of `x` to `2000`. Note that there may be one or more number of blank characters around =, but there is no tab characters.

3. `Goto <label>`, or `Goto <id> <label>`

   `Goto` is a keyword specifying an unconditional or conditional goto statement. label is a program line's number. For example `Goto 5` transfers the program execution flow to line 5 of the program, and `Goto x 5` when executed correctly, trasfers the flow to line 5 iff `x>0`, and to the next line otherwise. The label is guaranteed to be in the range of program line numbers.

4. `Inc <id>`, or `Dec <id>`

   Inc and Dec are keywords specifying increment and decrement statements respectively. For example `Inc x` (`Dec y`) when executed correctly adds (substracts) 1 to (from) the value of `x` (`y`).

5. `End`

   End is a keyword specifying the end statement, whose execution stops the program.

Not that the keywords of the DDL language are case-insensitive.

Error conditions:
When one of the following erroneous statements encounters during the program execution, an error message appears in a separate line of the output. Each error message is of the form <*label*> <*space*> <*error code*>. *label* is the line number for the erroneous statement, *space* is one blank character, and *error code* is a positive integer specified below.

1. `Dcl x` is erroneous if `x` has not been referenced (used in assignment, goto, increment or decrement) since the last time a `Dcl x` (declaring the same variable) statement has been executed, unless this is the first `Dcl x` statement being executed. In this erroneous condition, an error message indicating a repeated declaration is generated as <*label*> 1, where label is the program line number for the erroneous statement. Then the program flow transfers to the statement in the next program line, and any prior correctly executed declaration for `x` is valid.

2. Any other statement where a variable such as `x` is referenced (used in assignment, goto, increment or decrement) is erroneous if no `Dcl x` has been previously correctly executed. In this case, an error message indicating an undeclared reference is generated as <*label*> 2 and the program execution continues from the next line.

Input (filename: C.IN)

First line of the input file contains a single integer N indicating the number of DDL programs to follow ($1 \le N \le 20$). The first line of each test case contains a single integer indicating number of statements in that program which is in the range (1…100). There are no blank lines between test cases. Statements of each DDL program come one after the other in separate lines without any blank lines in between. Statements are not explicitly labeled, but they are implicitly labeled by the number of their line beginning from 1 for the first statement in each program. There is no syntax error in programs and they are guaranteed to terminate, and no overflow or underflow errors will occur during execution. In each line of the program, tokens (e.g. GOTO, =, etc.) are separated by at least one blank character. Also there may be some blank characters in the beginning or at the end of each line.

Output (filename: C.OUT)

For each input DDL program, your output should start with the program number in the first line, followed by the error messages generated by the program in the order they are generated, each error message in one line. There should be no blank lines between error messages.

Sample Input

```
2
4
DCL X
INC X
DCL X
END
9
DCL X
INC X
GOTO X 5
DCL Y
Y = 100
DCL X
DCL X
Y = 50
END
```

Sample Output

```
1
2
5 2
7 1
8 2
```

Problem D

(Program filename: D.CPP or D.PAS)

Color Tunnels

A company producing toys has a complex system to paint its products. To obtain the desired color, the product must be painted by several colors in a specified order. A product is painted by moving through color tunnels. For each color there is at least one tunnel that paints with that color, but there may be more. The tunnels are distributed in the painting area and the product must be delivered from one tunnel to another in order to be painted with the given colors. The product is at a certain point in the production plant when painting process starts and must finally be delivered to the product warehouse.

More formally, a finished uncolored product is at a certain given point (source point) and must be delivered to another given point (destination point) after being painted with different colors in a given order. There are several tunnels, each is assumed to be a line segment in the plain with a specific color. The colors of the tunnels are not necessarily distinct. Let $<c_1, c_2, \ldots, c_n>$ be the sequence of n colors that the product is to be painted with. The product is required to pass through tunnels $<t_1, t_2, \ldots t_n>$ such that the color of t_i is c_i. Note that it is possible to pass through a tunnel without being painted, so the mentioned $<t_1, t_2, \ldots t_n>$ may be in fact a subsequence of the tunnels which the product passes through. The direction in which the product passes a tunnel is not important. The goal is to find the shortest path from source to destination subject to the color constraints. The path may cross itself, or even cross a tunnel. Passing twice (or more) through a tunnel is also allowed. Note that two tunnels can cross or overlap but are different.

Input (filename: D.IN)

The input file contains several test cases. The first line of the input consists of a single integer t (between 1 and 20), the number of test cases. Following the first line is the data for t test cases. The first line of each test case contains four real numbers x_s, y_s, x_t, y_t which are x and y coordinates of the source and destination respectively. The second line of the test case contains the color sequence: the first number is the length of the sequence (between 1 and 30), and the rest of the line is the sequence itself. Each color in the sequence is an integer in the range 1…100. The third line contains a single integer n in the range 1…60 which is the number of tunnels followed by n lines each containing five numbers. The first two numbers are the x and y coordinates of one end of the tunnel. The third and fourth numbers are the x and y coordinates of the other end. Coordinates are real numbers. The fifth number is an integer in the range 1...100 representing the color of the tunnel.

Output (filename: D.OUT)

The output file must have t lines, each containing the minimum length of a path from source to destination subject to the constraints of the problem. The length will be compared to optimal length within a precision of three digits after decimal point.

Sample Input

```
1
0 1.5 100 67
4 1 4 3 1
9
10 10 20 20 1
10 15 20.5 35.333 3
30 15 14.55 12.5 1
40 30 44 33 1
29 84 33 58 4
9 39 41 115 2
75 47 37 69 4
46 26 58 25 3
73 48 27 59 3
```

Sample Output

```
240.60967918717043
```

Problem E

(Program filename: E.CPP or E.PAS)

Lazy Math Instructor

A math instructor is too lazy to grade a question in the exam papers in which students are supposed to produce a complicated formula for the question asked. Students may write correct answers in different forms which makes grading very hard. So, the instructor needs help from computer programmers and you can help.

You are to write a program to read different formulas and determine whether or not they are arithmetically equivalent.

Input (filename: E.IN)

The first line of the input contains an integer N ($1 \le N \le 20$) that is the number of test cases. Following the first line, there are two lines for each test case. A test case consists of two arithmetic expressions, each on a separate line with at most 80 characters. There is no blank line in the input. An expression contains one or more of the following:

- Single letter variables (case insensitive).
- Single digit numbers.
- Matched left and right parentheses.
- Binary operators +, - and * which are used for addition, subtraction and multiplication respectively.
- Arbitrary number of blank or tab characters between above tokens.

Note: Expressions are syntactically correct and evaluated from left to right with equal precedence (priority) for all operators. The coefficients and exponents of the variables are guaranteed to fit in 16-bit integers.

Output (filename: E.OUT)

Your program must produce one line for each test case. If input expressions for each test data are arithmetically equivalent, "YES", otherwise "NO" must be printed as the output of the program. Output should be all in upper-case characters.

Sample Input

```
3
(a+b-c)*2
(a+a)+(b*2)-(3*c)+c
a*2-(a+c)+((a+c+e)*2)
3*a+c+(2*e)
(a-b)*(a-b)
(a*a)-(2*a*b)-(b*b)
```

Sample Output

```
YES
YES
NO
```

Problem F

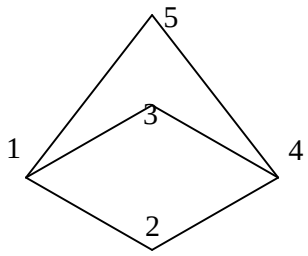
(Program filename: F.CPP or F.PAS)

Buggy Sat

Discovery Co. ltd. builds a satellite using a new kind of an intelligent camera. The camera has special software to detect cities and roads from an image, and is also able to detect every region (which is a connected part of surface), bounded by a series of connected roads with no other region inside. Using this technology, the satellite is able to compress the picture before sending it. The compressed format of a picture is the city locations and its regions.

Discovery has launched the satellite, without testing the software completely. So, after a while, they received some buggy pictures which includes one more extra region: the outer region. The outer region is the region of the plane enclosing every other region (which has infinite area). Further analysis shows that all images sent have the following properties:

1. All cities, in the image, have at least two roads to the other cities.
2. There is a path connecting every pair of cities.
3. There is at most one road between each pair of cities.
4. Roads do not cross each other except at the cities.



The above Figure shows a sample image received (see sample input).
You are to write a program to read a buggy image and report the outer region.

Input (filename: F.IN)

The first line of the input consists of a single integer N ($1 \le N \le 20$), which is the number of test cases. The test cases appear with no blank lines in between. The first line of each test case consists of the number of cities (between 1 and 50) followed by pairs of integers (x, y) which are location of cities (each pair in one line), followed by number of faces in a separate line (between 1 and 50), followed by face information on each line. Face information consists of number of cities making the face and the city numbers in clockwise (or counterclockwise) order.

Output (filename: F.OUT)

There should be a single line containing the boundary face number for each test case, with no blank lines in between.

Sample Input

```
1
5
2 6
4 4
4 7
8 6
4 10
3
4 1 2 4 3
4 1 3 4 5
4 1 2 4 5
```

Sample Output

```
3
```

Problem G

(Program filename: G.CPP or G.PAS)

Cashier Employment

A supermarket in Tehran is open 24 hours a day every day and needs a number of cashiers to fit its need. The supermarket manager has hired you to help him, solve his problem. The problem is that the supermarket needs different number of cashiers at different times of each day (for example, a few cashiers after midnight, and many in the afternoon) to provide good service to its customers, and he wants to hire the least number of cashiers for this job.

The manager has provided you with the least number of cashiers needed for every one-hour slot of the day. This data is given as $R(0)$, $R(1)$, …, $R(23)$: $R(0)$ represents the least number of cashiers needed from midnight to 1:00 A.M., $R(1)$ shows this number for duration of 1:00 A.M. to 2:00 A.M., and so on. Note that these numbers are the same every day. There are N qualified applicants for this job. Each applicant i works non-stop once each 24 hours in a shift of exactly 8 hours starting from a specified hour, say t_i ($0 \le t_i \le 23$), exactly from the start of the hour mentioned. That is, if the ith applicant is hired, he/she will work starting from t_i o'clock sharp for 8 hours. Cashiers do not replace one another and work exactly as scheduled, and there are enough cash registers and counters for those who are hired.

You are to write a program to read the $R(i)$'s for i=0..23 and t_i 's for i=1..N that are all, non-negative integer numbers and compute the least number of cashiers needed to be employed to meet the mentioned constraints. Note that there can be more cashiers than the least number needed for a specific slot.

Input (filename: G.IN)

The first line of input is the number of test cases for this problem (at most 20). Each test case starts with 24 integer numbers representing the $R(0)$, $R(1)$, …, $R(23)$ in one line ($R(i)$ can be at most 1000). Then there is N, number of applicants in another line ($0 \le N \le 1000$), after which come N lines each containing one t_i ($0 \le t_i \le 23$). There are no blank lines between test cases.

Output (filename: G.OUT)

For each test case, the output should be written in one line, which is the least number of cashiers needed.
If there is no solution for the test case, you should write `No Solution` for that case.

Sample Input

```
1
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
5
0
23
22
1
10
```

Sample Output

```
1
```

Problem H
(Program filename: H.CPP or H.PAS)

Dolphin Pool

In a newly constructed dolphin pool in the Kish island in Persian Gulf, one of the fun games is as follows: the game director throws several plastic rings in the pool such that center of no ring lies inside any other ring, and no two rings are tangent. The dolphins are trained to jump out on the director's whistle through the closed areas that are completely outside the rings, one dolphin from one such area. The dolphins jump out if and only if the number of closed areas exactly equals to the number of dolphins.

You are to write a program to given the following input/output description, finds the number of closed areas between rings to help the dolphins decide to jump out or not.

Input (filename: H.IN)

The first line includes the number of test cases (at most 20). Each test case data has an integer N ($1 \le N \le 20$), the number of plastic rings, in its first line. Following the first line there are N lines, each containing three integers, the first and second being the x and y coordinates of the circle of the ring, and the third is its radius. Coordinates are positive integers less than 1000 and the radius is in the range 1…100.

Output (filename: H.OUT)

For each test case, there must be one line in the output including the number of closed areas in that test case.

Sample Input

```
2
4
100 100 20
100 135 20
135 100 20
135 135 20
1
10 10 40
```

Sample Output

```
1
0
```

Problem I

(Program filename: I.CPP or I.PAS)

3002 Rubbery

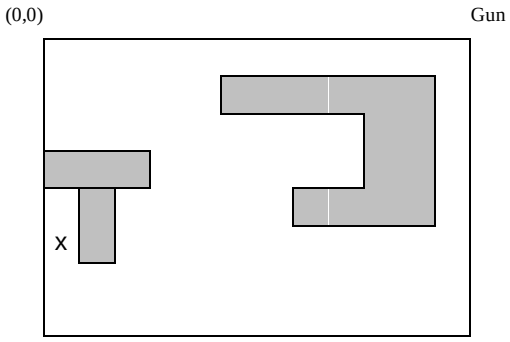
It's now 3002 and the Godfather *Unlucky Luchiano* is planning a rubbery from the Rectilinia museum. The problem is that the walls of the museum are impenetrable and the doors are guarded so his men cannot enter the museum from its doors. He is lucky that the museum has no roofs and one can enter the museum from the above. So he decides to use a device from thousands of years ago; a catapult! Using this, his men can fly and fall in some place in the museum without being caught by guards.

But another problem still exists which is a high-tech laser gun that guards the museum. The laser gun is designed based on a very recent discovery that it is possible to guide the laser beam not on a straight line, but on a rectilinear path. The fact that the beam travels the shortest (possible) distance from the emission point to a target still remains. The problem is that if one land somewhere in the museum that is reachable by the laser beams, he will be immediately destroyed. Again godfather is lucky, since in the museum, there are some walls and other obstacles through which the beam cannot pass and if his men can land in the shadows of that walls, they are safe in the way that they are not destroyed by the gun right after landing and they can use their special device to disable the laser gun. As the catapult is not a very precise device, godfather wants to know the probability of landing in shadows, so he has to compute the total area of shadowy regions of the museum.

Given the layout of the museum, you have to write a program to compute the total area of the shadowy regions. Yes! This time you really HAVE TO!

As you study the layout of the museum from the top view, you find that the museum can be considered as a rectangle with some obstacles in it. The obstacles are simple polygons with sides parallel to the rectangle sides. Interior of obstacles do not overlap. The laser gun is located in the upper-right corner of the museum. A laser beam is composed of a number of line segments; each is either horizontal or vertical. When the gun chooses its target, it intelligently determines a possible path to the target and fires. A possible path has the following characteristics:

- It consists of only horizontal and vertical segments.
- It never crosses an obstacle but in some parts may be tangent to obstacle sides. It can never be tangent to two obstacle sides at one point.
- In the travel from the gun to the target, the beam never moves from left to right or from bottom to top (directions are relative to view of the museum from the above).



The problem is to compute the total area of the shadowy regions in the museum (not belonging to the interior of the obstacles), where the laser gun cannot shoot any point inside those regions. In the above figure, the point marked by (x) is in a shadowy region (see sample input).

Input (filename: I.IN)

The input file contains several test cases. The first line contains a single integer *t* (between 1 and 10), which is the number of test cases. Rest of the input file contains *t* test cases. The first line of each test case contains 2 positive integers, which are the length and the width of the rectangle respectively. The second line contains a single integer *n*, which is the number of obstacles in the museum (between 0 and 50). After it, there are *n* lines, which contain the obstacle data. Each obstacle data is in a single line, which begins with an integer *m*, which is the number of vertices of the obstacle (between 4 and 50), followed by 2*m* numbers, which are the x, and y coordinates of the vertices listed in clockwise order. Every coordinate in the input is a positive integer less than 1,000,000. Upper-left corner of the rectangle is the coordinate origin.

Output (filename: I.OUT)

The output file must contain exactly *t* lines. Each line contains a single number, which is the total area of the safe regions in the test case. The output for each test case is guaranteed to fit in a 32-bit integer. Total execution time for all test cases must be less than 1 minute.

Sample Input

```
1
12 8
3
8 5 1 11 1 11 5 7 5 7 4 9 4 9 2 5 2
4 0 3 3 3 3 4 0 4
4 1 4 2 4 2 6 1 6
```

Sample Output

```
12
```