Problem A
Load Balancing

(Time limit: 1 second)

In a network of m identical servers, n requests are sent to these servers for services. Because some requests take a large number of resources, we do not want two or more of such requests to be assigned to the same server. This kind of request management is called load balancing. Because the cost of the servers is expensive, we want to use as few servers as possible. Given n requests $\{1, 2, 3, \ldots, n\}$ and p pairs $\{\{i, j\} \mid i \neq j\}$ of requests that need to be assigned to different servers, your job is to assign m servers to these requests such that m is as small as possible.

Input

There are two test cases in the input file. The first line contains n and p for the first test case, where $2 \leq n \leq 7$ and $1 \leq p < 25$. The second line contains $2p$ integers representing the p pairs $\{\{i, j\} \mid i \neq j\}$ of the first case. The third line specifies the input for the second case in the same fashion.

Output

In the output file the first line contains m_1 and m_2, where m_1 is the smallest number of the first case and m_2 the second case. The two numbers are separated by a blank.

Example

Input	**Output**
2 1 1 2 4 5 1 2 3 1 1 4 3 4 2 3	2 3

Problem B
Pecking Chicks
(Time limit: 1 second)

In a community of n chicks it is no secret that chicks like to peck one another. Let $C = \{1, 2, 3, \ldots, n\}$ be a community of n chicks. We say that chick i *virtually pecks* chick j if

1. Chick i pecks chick j, or
2. Chick i pecks chick k and chick k *virtually pecks* chick j.

Observe that if chick i pecks chick j, it is not necessary that chick j pecks chick i. Assume also that chick i pecks itself. Given all pairs of pecking chicks, design an efficient algorithm and write a program to find every pair of chicks that virtually peck the other in the pair.

Let <*i*, *j*> denote chick i pecks chick j. For example, you are given a set $C = \{1, 2, 3, 4\}$ and four pairs of pecking chicks <1, 2>, <2,1>, <1, 3>, <2, 3>, and <3, 4>. The solution is <1, 1>, <1, 2>, <1, 3>, <1, 4>, <2, 1>, <2, 2>, <2, 3>, <2, 4>, <3, 3>, <3, 4>, <4, 4>.

Input

The first line of the input file contains the number t of test cases, where $2 \le t \le 7$. The second line contains the number n of chicks for the first test case, where $2 \le n \le 1{,}000$. The third line contains $2m$ integers, representing m pairs of pecking chicks. The fourth line starts over for the second test case in the same fashion.

Output

The output file contains t lines of integers. Line one represents k_1 pairs of chicks that virtually peck the other in the pair for the test case 1. Line two represents k_2 pairs of chicks that virtually peck the other in the pair for the test case 2 and so on. Note that for each test case your program must report a solution in lexicographic order, that is, by sorting the first element of the pairs, then by the second.

Example

Input	**Output**
2 2 1 2 4 1 2 2 1 1 3 2 3 3 4	1 1 1 2 2 2 1 1 1 2 1 3 1 4 2 1 2 2 2 3 2 4 3 3 3 4 4 4

Remark: The second test case in the input illustrates the example in the problem.

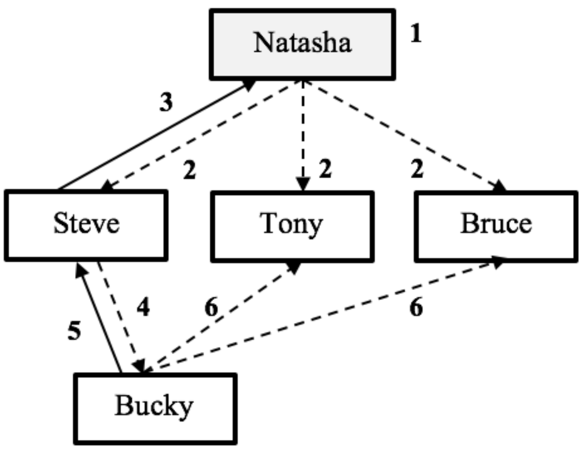
Problem C
Babyslam's Lost Tickets

(Time limit: 1 second)

Babyslam, one of the most favorite rock bands of all time in Thailand, is throwing the special private concert in Chiang Mai province. Since the tour is private and special, the *online* concert tickets will be *very limited* and must be *invited only*.

To get a ticket, it is important that *you must either be one of the exclusive fans* of the band or *at least know one of the fans*. The exclusive fans are those who register in advance for the concert event. Thus, they are the only tier that will get invited by the band. But this does not mean that all exclusive fans will be ticketed: there are limitedly *n* tickets available for the first *n* persons who make a register.

After being ticketed, each individual possessing the ticket can decide whether to *keep it* or *give it to a number of selected persons he/she knows*. The first who in turn claims for the ticket will be ticketed instead of the ticket giver. Every person can possess only at most one ticket. Note that it is also possible for a person to claim the ticket and then later give it to some others. However, tickets cannot be claimed from those who are already a ticket giver.



For example, the above figure illustrates the ticketing situation that comprises the following events that occur in chronological order:

1) Natasha is an exclusive fan.
2) Natasha decided to give her ticket to one among Steve, Tony, and Bruce.
3) Steve claimed for the ticket from Natasha. So, he was ticketed instead of Natasha.
4) Steve decided to give his ticket to Bucky.
5) Bucky in turn claimed for the ticket. Bucky was ticketed instead of Steve.
6) Bucky decided to give his ticket to Tony or Bruce.

Observe that in the above situation the ticket originally from Natasha would be considered as *a lost ticket* if there will be no claim neither from Tony nor Bruce. Also, note that it is possible for Steve to claim for the ticket from Natasha before she decides to give her ticket.

As Babyslam is highly popular in Chiang Mai, one lost ticket is good for no one. Indeed, what the band has in mind is that all lost tickets should be re-ticketed to the rest of their exclusive fans. It is your job now to help them write a program that can check for the number of lost tickets. The input and output of the program are described in what follows.

Input

The first line of the input file contains the number of test cases t. Each test case comprises m+2 lines, where $1 \leq m \leq 10{,}000$. For each test case, the first line contains the names of n exclusive fans (those who complete a register), separated by space, where $1 \leq n \leq 400$. The second line contains integer m. This is followed by m lines each of which contains the information of a ticket event that input in chronological order. Each event comes in two flavors: (G) given and (C) claim. The given event is input by the letter G, followed by k strings $N_0\ N_1 \ldots N_k$, $2 \leq k \leq 10$, separated by space, meaning that person N_0 decides to give his/her ticket to other k persons $N_1, N_2, \ldots, N_k$. The claim event is input by the letter C, followed by *2* strings $C_0\ C_1$ separated by space, indicating that person C_0 claims for the ticket from C_1. Remark that your program should be able to handle the load up to 5,000 users.

Output

The output contains t lines. Each line contains the number of lost tickets in each test case, respectively.

Example

Input	**Output**
3 Erik Charles 4 C Jean Erik G Erik Jean G Charles Logan C Logan Charles Natasha 5 G Natasha Steve Tony Bruce C Steve Natasha G Steve Bucky C Bucky Steve G Bucky Tony Bruce Scott Hank 1 G Scott Hank	0 1 1

Remark: The second test case in the input illustrates the example in the problem.

Problem D
I Love Natural Numbers
(Time limit: 1 second)

A natural number is the number in the set {0, 1, 2, 3,…}. Given a number *n*, your job is to determine whether *n* is a natural number.

Input
The first line of the input file contains an integer $t \leq 1000$, representing the numbers to be tested. The next line contains *t* numbers. Each number is separated by a blank space.

Output
The output has a sequence of *t* characters in the set {Y,N}. If the first number is a natural number, your program should output Y or N, otherwise. For the following *t*-1 numbers, your program should output Y or N in the same way.

Example

Input	Output
4 5.3 4 6.2 7.9	NYNN

Problem E
Bounding Box
(Time limit: 1 second)

For some reasons, we want to draw certain geometrical shapes on a canvas which is referenced by the two dimensional Cartesian coordinate system. The coordinate system has a horizontal axis (x-axis) and a vertical axis (y-axis) whose values are in the range of -100.0 to 100.0.

We need to calculate the total "**bounding box**" which is the smallest rectangle that is just big enough to hold all the shapes drawn on the canvas. A bounding box is specified by the coordinate of "**top-left corner**" and the coordinate of "**bottom-right corner**".

Assume that there are only three kinds of shapes: circle, rectangle, and line segment. We can specify the **location** and the **size** of each shape using the following patterns:

- **Circle**: '**c**' character, (x, y) coordinate of the center, length of the radius
- **Rectangle**: '**r**' character, (x, y) coordinate of the top-left corner, width, height
- **Line Segment**: '**l**' character, (x_1, y_1) coordinate of the starting point, (x_2, y_2) coordinate of the ending point

Write a computer program to help calculating a total bounding box of a given list random geometrical shapes.

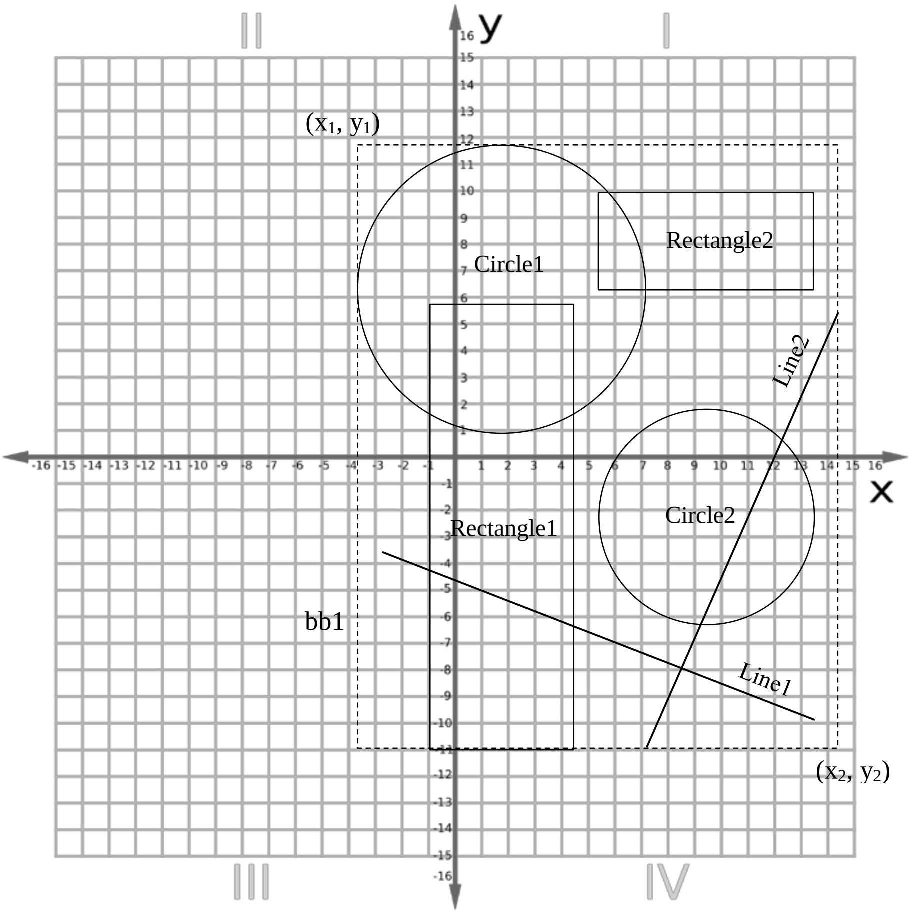


Figure: Total bounding box “**bb1**” specified by “**(x_1, y_1), (x_2, y_2)**”

Input:

- The first line is the number *t* of cases to be calculated, where $1 \le t \le 10$.
- The following lines are the list of random shapes. Each line contains a single shape specified according to the patterns above. All parameters of each shape are “space” separated.
    - All coordinate and dimension values can be the floating point type with one decimal place (e.g. 25.5, 0.0 or -33.7).
    - ‘0’ indicates the end of each case.

Output:

- The coordinate of the **top-left corner** followed by coordinate of the **bottom-right corner** of the total bounding box **on the separated lines**.
- The resulting coordinates have to be floating point with one decimal place.

Input 1	Output 1
3 c -2 2 4 c 0 2 5 c 2 -2 4 0 c 0 2.0 5.0 c 0 -2.0 7 r 0 0 8 8 r -6 -6 4.0 3 0 c 0 0 0 0	-6.0,7.0 6.0,-6.0 -7.0,7.0 8.0,-9.0 0.0,0.0 0.0,0.0
Input 2	**Output 2**
3 c 0 0 4.5 l -5.5 -3.5 0 0 l 0 0 5.5 3.5 0 l 4 4 4 -4 l 4 -4 4 4 0 l -10.8 0 0 0 l 10.8 0 0 0 l -5 8.4 -5 -8.4 0	-5.5,4.5 5.5,-4.5 4.0,4.0 4.0,-4.0 -10.8,8.4 10.8,-8.4

Problem F
Spearman's Rank-Order Correlation
(Time limit: 1 second)

The Spearman correlation coefficient, *scc*, between two ranks of sequences ranges from +1 to -1. An *scc* at +1 indicates a complete association between such two ranks. The -1 indicates negative association completely between the ranks. Meanwhile the 0 of *scc* indicates no association between ranks. The *scc* in the case of all the data is distinct between two sequences with an equal size, $s1 = \langle s1_1, s1_2, s1_3, \ldots s1_n \rangle$ and $s2 = \langle s2_1, s2_2, s2_3, \ldots s2_n \rangle$ can be computed as follows.

$$scc = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}$$

where $d_i = \text{rank}(s1_i) - \text{rank}(s2_i)$, $\text{rank}(sx_i)$ is the ranking of sx_i in *sx* in descending order. That is, the largest number in the sequence has the rank 1, the second largest has the rank 2, and so on. The smallest number has the rank *n*.

For example, given *s1* = *<56, 75, 45, 71, 62, 64, 58, 80, 76, 61> and s2 =<66, 70, 40, 60, 65, 56, 59, 77, 67, 63>*, the *scc* is 0.66.

Given two such sequences, determine whether the Spearman correlation coefficient is Positive, Negative, or Zero.

Input:

m pairs of sequences, $1 \le m \le 5$. Each pair contains two sequences of size $1 \le n \le 10$, one sequence per one line. The first and second lines in a pair is the first and second sequences, respectively. The assumption is that, all the data within any sequence is distinct, and each pair of sequences always has the same size.

Output:

There are *m* lines of output, each line of the output contains Positive, Negative, or Zero result of the Spearman correlation coefficient.

Sample Input	Sample Output
56, 75, 45, 71, 62, 64, 58, 80, 76, 61 66, 70, 40, 60, 65, 56, 59, 77, 67, 63 1 2 3 1 2 3 10 20 30 40 50 50 40 30 20 10	Positive Positive Negative

Problem G
Alien Alphabet Order
(Time limit: 1 second)

You are a language scientist or a linguist at NASA. Currently, you are in a secret mission because NASA just made the first contact with an alien race. Well, as you have seen from many movies, there is a higher chance to have a big war with them. So, your job is to avoid the war by telling them "don't fight, let's be friend". However, we do not know their language. What NASA has is a sequence of signals which is represented by a sequence of English characters for simplicity.

Your job is to find the ordering of characters (i.e., the collating sequence) in alien language. In particular, your program will take a set of strings that has been sorted according to a particular collating sequence and determine what that sequence is.

Input

The input consists of a multiple set of t test cases, $1 \leq t \leq 10$. The first line will be an integer t indicating the number of test cases. Each test case consists of an ordered list of strings of uppercase letters, one string per line. Each string contains at most 100 characters. The end of the list is determined by a line that contains a single character '#'. The list size is at most 100 strings. The list will imply a complete ordering among those letters that are used. A sample input file is shown.

Output

For each test case, your output should be a single line containing uppercase letters in the order that specifies the collating sequence used in producing the input data file. Correct output for the data file above is shown below.

<table>
<tr><th>Sample Input</th><th>Sample Output</th></tr>
<tr><td><pre>2
XWY
ZX
ZXY
ZXW
YWWX
#
ZX
ZW
XW
#</pre></td><td><pre>XZYW
ZXW</pre></td></tr>
</table>

Problem H
Maximum Sum
(Time limit: 1 second)

ให้ตัวเลขจำนวนเต็มมา n x n ตัว โดยที่ $1 \leq n \leq 100$ กำหนดให้ตัวเลขจำนวนเต็มแต่ละตัวคือ a_{ij} มีค่าระหว่าง -127 ถึง 127 ซึ่ง i คือแถว $0 \leq i \leq n-1$ และ j คือ คอลัมน์ $0 \leq j \leq n-1$ โจทย์คือ ให้เขียนโปรแกรมหาตำแหน่งของสี่เหลี่ยมที่มีผลรวมของตัวเลขจำนวนเต็มทุกตัวภายในสี่เหลี่ยมที่มีค่ามากที่สุด โดยบอกตำแหน่งในลักษณะ "(แถว, คอลลัมน์)" ประกอบด้วยมุมซ้ายบน (m, l) มุมขวาบน (m, r) มุมซ้ายล่าง (n, l) มุมขวาล่าง (n, r)

<u>หมายเหตุ</u> ในกรณีที่มีคำตอบได้หลายคำตอบ ให้เลือกคำตอบของขอบเขตสี่เหลี่ยมที่มีมุมซ้ายบน (m, l) ที่ต่ำที่สุด (นั่นคือ มีค่า m น้อยที่สุดและค่า l น้อยที่สุด) เป็นคำตอบดังตัวอย่างที่ 3

ข้อมูลเข้า(Input)

บรรทัดแรกบอกจำนวนกรณีทดสอบ (test cases) $1 \leq t \leq 13$, แต่ละกรณีทดสอบประกอบด้วย บรรทัดแรกคือ ตัวเลขจำนวนเต็ม n บอกจำนวนแถวและคอลัมน์ ซึ่งจะบอกว่ามีตัวเลขจำนวนเต็มจำนวน n x n ตัว บรรทัดถัดมาจำนวน n บรรทัดคือ ตัวเลขจำนวนเต็ม a_{ij} โดยที่ i บอกตำแหน่งแถว $0 \leq i \leq n-1$ และ j คอลลัมน์ $0 \leq j \leq n-1$ โดยตัวเลขในแถวเดียวกันแต่ละตัวจะขั้นด้วยช่องว่าง (Space)

ข้อมูลออก(Output)

ข้อมูลออกจะมีทั้งหมด t บรรทัด แต่ละบรรทัดแสดงข้อมูลของแต่ละกรณีทดสอบตามลำดับประกอบด้วย ผลรวมสูงสุด และแสดงตำแหน่งของสี่เหลี่ยมที่ให้ผลรวมมากที่สุด (แถว, คอลลัมน์) ประกอบด้วยมุมซ้ายบน (m, l) มุมขวาบน (m, r) มุมซ้ายล่าง (n, l) มุมขวาล่าง (n, r) ทุกข้อมูลขั้นด้วยช่องว่าง (Space)

<u>ตัวอย่าง</u>

ข้อมูลเข้า	**ข้อมูลออก**
3 4 0 -2 -7 0 10 3 -6 2 -4 1 -4 1 -1 8 0 2 2 -2 -3 -1 -5 3 1 6 -2 -2 -2 -3 -2 -2 7	17 (1, 0) (1, 1) (3, 0) (3, 1) -1 (1, 0) (1, 0) (1, 0) (1, 0) 7 (0, 0) (0, 1) (0, 0) (0, 1)

Problem I
The Table

(Time limit: 1 second)

You are given a table of size x rows by y columns, and two numbers *diff* and *num*. Each alternating position in the table has a number assigned to it ordering from left to right and top to bottom. Each of the $x \times y$ squares in the table is identified by its row and column indices. The indices always start from 0 so that the first position in the table is indicated by (0, 0) (See Figures below). Some small squares in the table will be filled with numbers according to the following rules.

- The first number assigned to the position (0, 0) is always 1.
- The second number 1+*diff* is assigned to (0, 2), the third number 1+2*diff* is assigned to (0, 4), the fourth number 1+3*diff* is assigned to (0, 6), and so on from left to right and top to bottom until the table cannot be filled. See examples below.

Example 1: A fully assigned table of size 9 x 10 with *diff* = 2.

	0	1	2	3	4	5	6	7	8	9	Column
0	1		3		5		7		9		
1		11		13		15		17		19	
2	21		23		25		27		29		
3		31		33		35		37		39	
4	41		43		45		47		49		
5		51		53		55		57		59	
6	61		63		65		67		69		
7		71		73		75		77		79	
8	81		83		85		87		89		

Row

Example 2: A fully assigned table of size 5 x 5 with *diff* = 3.

Column

	0	1	2	3	4
0	1		4		7
1		10		13	
2	16		19		22
3		25		28	
4	31		34		37

Row

Observe that *num* is guaranteed to be in the table. Given a number *num* and a fully assigned table, your job is to find *num*'s "neighbors". The number *num* in the table may have up to 4 neighbors. These neighbors are in the 4 directions of *num* : north-west (NW), north-east (NE), south-west (SW), and south-east (SE). In the example 1, if *num* is 45, the 4 neighbors of 45 are 33 (NW), 35 (NE), 53 (SW), and 55 (SE). In the example 2, if *num* is 16, the 2 neighbors of 16 are 10 (NE) and 25 (SE). Note that 16 has no neighbors in the NW and SW directions.

Please help us find the coordinate of *num* and its four neighbors.

Input

In the input file, the first line contains the number t of test cases, where $1 \leq t \leq 25$. The second line contains four integers representing *num*, *diff*, x, and y, respectively, of the first case. The third line contains four integers representing *num*, *diff*, x, and y, respectively, of the second test case. The fourth line repeats in the same fashion. In all cases, $1 \leq num \leq 4{,}290{,}000{,}000$, $1 \leq diff \leq 20$, $5 \leq x \leq 45{,}000$, and $5 \leq y \leq 45{,}000$.

Output

In the output file, the solution to each test case contains exactly 4 lines. The first line reports *num*'s neighbor in the NW direction. The second line reports *num*'s neighbor in the NE direction. The third line reports *num*'s neighbor in the SW direction. The fourth line reports *num*'s neighbor in the SE direction. If *num*'s neighbor does not exist, report -1. You are asked to report the solution to the first test case first, then the solution to the second test case, and then the solution to the third test case, and so on.

Example

Input	Output
2 45 2 9 10 16 3 5 5	33 35 53 55 -1 10 -1 25