13th México and Central America
ACM International Collegiate Programming Contest 2008
Problem No. 1 – Trip to Mars
Input file: trip.in **Output file:** Standard Output

In the year 2048 the IT community in Mars is planning to send exploration trips to find new galaxies. In order to be eligible, astronauts need to be member of the Galactic Training Program (GTP). In order to take a training course the astronaut must complain with two requirements. First the astronaut must at least have 180 days as member of the GTP, and he must have an “A” (Alfa) certification. If both requirements are met the astronaut is accepted, in other cases the astronaut is not accepted to take the training course. Your task is to read the astronauts information and determine which astronauts are accepted and assign them a number starting in 1. Do these for each group and simulation date provided. Dates are in the format DD/MM/YY. Remember that February is a special month with 28 days and ounces every four years it has 29 days.

Input
The first line of the input defines how many simulations “n” you need to run. Following this you will find “n” groups, the first element being the date of the simulation follow by the list of astronauts. For each astronaut you will find the date of enrollment to the GTP, his name and his certification. Certifications are represented with one alphabet character and astronauts only have one certification at any time. The three elements of each line are separated by a space.

Output
For each simulation display a blank line, the date in the next line and one line for each astronaut indicating if he is accepted (or denied) and his acceptance number.

Sample Input
```
2
25/09/08
12/02/08 Juan Jiménez A
06/08/08 Martha Martínez B
09/03/08 Raúl Ramírez A
03/10/08
20/08/08 Maria González A
```

Sample Output
```
Current date 25/09/08
Accepted Juan Jiménez 1
Denied Martha Martínez
Accepted Raúl Ramírez 2

Current date 03/10/08
Denied Maria González
```

13th México and Central America

ACM International Collegiate Programming Contest 2008
Problem No. 2 – Binary Clock
Input file: binary.in **Output file:** Standard Output

A **binary clock** is a clock which displays traditional sexagesimal time in a binary format. The most common binary clock uses six columns of LEDs to represent zeros and ones. Each column represents a single decimal digit, a format known as binary-coded decimal (BCD). The bottom row in each column represents 1 (or 20), with each row above representing higher powers of two, up to 23 (or 8). To read each individual digit in the time, the user adds the values that each illuminated LED represents, and then reads these from left to right. The first two columns represent the hour, the next two represent the minute, and the last two represent the second.

For example:

() = turn off
(*) = turn on

	H	H	M	M	S	S
2^3		()		()		(*)
2^2		()	()	(*)	(*)	()
2^1	()	()	(*)	(*)	()	()
2^0	(*)	()	(*)	(*)	()	(*)
	1	**0**	**3**	**7**	**4**	**9**

Time is:

10 **:** 37 **:** 49

Your task for this problem is simple: read an hour in its binary format and output its equivalent in sexagesimal time format.

In order to facilitate your task, each one of the six columns of LEDs than represents a single decimal digit is concatenated as shown below.

For example, 10:37:49 would be write as: ()(*)()()()()()(*)(*)()(*)(*)(*)(*)()()(*)()()(*)

Note: Spaces between characters do not exist.

Input

The input consists of multiple test cases. The first line of input contains a single integer N, (1 ≤ N ≤ 1000) which is the number of test cases that follow. Each test case consists of a single line containing a string that represents an hour in its binary format. The length of the LEDs string is guaranteed to be <= 54.

Output

For each test case, print the case number (beginning with 1) followed by the hour in its equivalent sexagesimal time format (HH:MM:SS). Follow the format shown in the sample output.

Sample Input	Samples Output

`2`
`()(*)()()()()()(*)(*)()(*)(*)(*)(*)()()(*)()()(*)`
`()()()()()()()()()()()()()()()()()()()()`
	`Case 1: 10:37:49`
	`Case 2: 00:00:00`

13th México and Central America
ACM International Collegiate Programming Contest 2008
Problem No. 3 – Chinese Checkers
Input file: chinese.in **Output file:** Standard Output

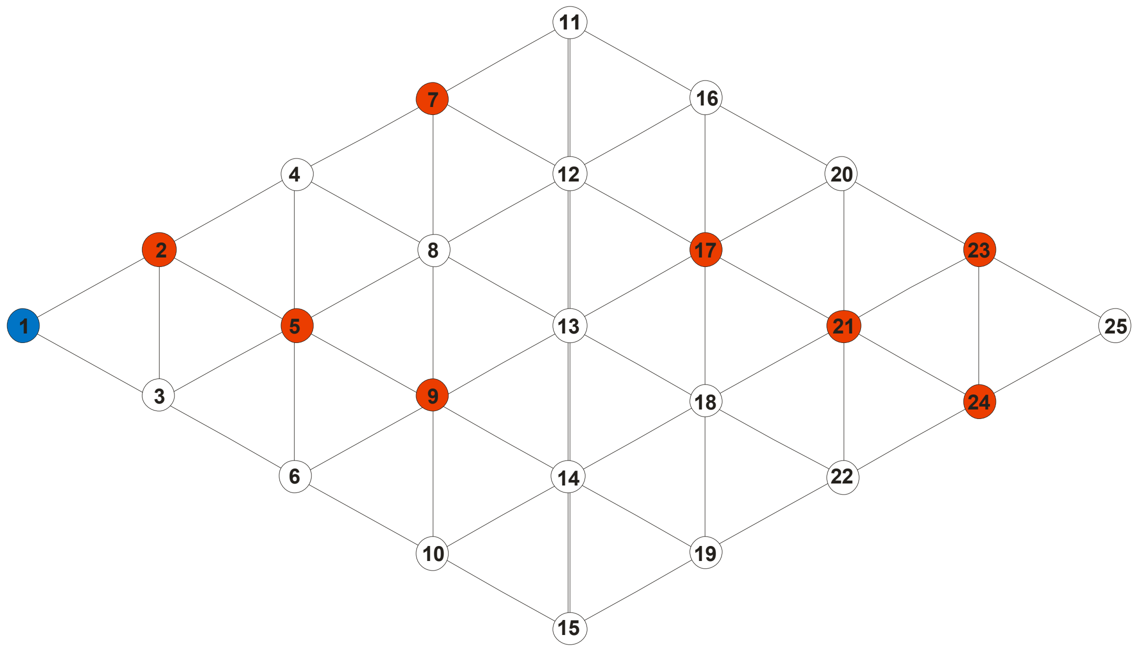
When playing Chinese Checkers, the objective of the game is to place one's marbles in the corner opposite of one's starting position of a pitted hexagram with form of a six-pointed star by single moves or jumps over other pieces.

The rules to move the marbles through the board are:

- You can move to any adjacent and unoccupied cell
- If there's an unoccupied cell after an adjacent occupied cell (through a straight line), you can jump it and repeat it as many times as you need.

A naive but common way to move the pieces on the board is selecting a marble and looking for the sequence of moves that reach as close to the destination corner as possible (the cell with the highest value). Your job will be to find this route.

Suppose that marbles are always in three possible places: the center of the board, the origin corner and the destination corner. So you have a scenario like this.



The problem to solve is for a scenario with a variable number of cells, determined by the number of cells in each side of the figure.

Input
The first line will contain the number of cases to solve. After this, for each case, you will have:

- N, the number of cells for each side of the scenario. 2≤N≤30.
- In the next line there is a list of occupied cells.
- In the next line there is a list of origin positions.

You can assume that none of the origin positions are in the list of occupied cells.

None line will be more than 2000 characters
There's a blank line to separate each test case

Output

Print Case #<number>: for each test case. In the next line print for each origin position the cell closest to the destination cell that can be reached. If there's more than one cell that has the same distance to the destination, print the one with the highest value. Let's define the distance to destination as the minimum number of moves from the origin to the destination cell ignoring occupied cells, for example 16, 17, 18 and 19 are 3 moves far from 25.
Print a blank line to separate the test cases.

The sample input corresponds to the above figure, with the first origin cell (in the list) corresponding to the blue marble in the figure. Note that this marble can move from 1 to 25 through the sequence 1 – 4 – 6 – 13 – 20 – 25.

Sample Input

```
1
5
2 5 9 7 17 21 23 24
1 11 8 16
```

Sample Output

```
Case #1:
25
25
13
20
```

13th México and Central America
ACM International Collegiate Programming Contest 2008
Problem No. 4 – Cycles and more cycles
Time Limit: 2 Seconds
Input file: cycles.in **Output file:** Standard Output

You are given a connected undirected graph G = (V,E) , and a list of special nodes. This list gives the order in which the special nodes must be visited.

Your task is to count the number of Hamiltonian cycles such that the special nodes are visited in order. To avoid confusions with the rotations the cycles always start at the first special node.

A Hamiltonian path is a path in an undirected graph which visits each vertex exactly once.

Input
The first line will be the number of test cases. For each test case you will receive n (2<=n<=15), the number of nodes. Then n lines with n numbers, where the j-th position of the i-th line is 1 if there is a connection between the i-th node and the j-th node, 0 if there is not connection. Then k (1<=k<=n) the number of special nodes, followed by k lines indicating the numbers of the special nodes.

Output
The number of Hamiltonian cycles asked in the description. As the result could be a large number print the result modulo 98765431.

Sample Input
```
2
4
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
2
1
2
2
0 1
1 0
1
0
```

Sample Output
```
5
1
```

Explanations:

First test case	Second test case

`1 0 3 2`
 	
`1 2 0 3`
 	
`1 2 3 0`
 	
`1 3 0 2`
 	
`1 3 2 0`
	`0 1`

13th México and Central America

ACM International Collegiate Programming Contest 2008
Problem No. 5 – Minimal Multiple
Time Limit: 4 Seconds
Input file: minimal.in **Output file:** Standard Output

John has a number that can be modified by changing the position of its digits. The rules are the following: if one digit is in position A it can only be changed to a position B if and only if |**A - B**| <= **k** (namely, only if the absolute difference of **A - B** is less or equal to one given **k**). John wants to know how many numbers can be generated that satisfy the property of being multiples of **M**.

Besides that, John he wants to know the minimal of number that satisfies the above restriction.

Input

Each input test case consists on the following:

- N (0 <= N <= 10^1000), k (0 <= k <= 3) and M (0 < M <=100).

Output

For each test case you must be print the number of numbers that you found modulo 10007 and the minimal number of them, otherwise print -1.

Sample Input	**Sample Output**

`12342132111111 0 33`
`12342132111111 0 9`
`12342132111111 3 3`
	`1 12342132111111`
	`0 -1`
	`5625 12122341131111`

13th México and Central America
ACM International Collegiate Programming Contest 2008
Problem No. 6 – Can I take it?
Input file: can.in **Output file:** Standard Output

Given a list of courses, which contain the name of the course, the points awarded by taking the course and the points needed before you can take the course, follow by a list of people trying to take courses. Determine whether a person can take a course or not, based on the points previously earn by each person.

Input
The input consists of: The number of runs. Follow by a list of courses, consisting of course name, the points awarded by the course and the points needed to take the course. Follow by a list of persons, each line with the person name, his points and the name of the course which the person desired to take. There is a blank line before the second list and a blank line after each run.

Output:
The list of the people with 'Accepted' or 'Refused' classification and the new credits after the course has been taken.

Sample Input
```
1
POO 10 0
ALGORITMS 10 0
DATA STRUCTURES 10 0
CVP 10 30
LINEAR ALGEBRA 10 30

Javier Jimenez 10 CVP
Arturo America 0 POO
Carlos Calzada 10 ALGORITHMS
Carlos Calzada 20 POO
Carlos Calzada 10 DATA STRUCTURES
Felipe Wong 0 ALGORITHMS
Gonzalo Alvarez 50 CVP
Maria Perez 30 CVP
Alma Garcia 20 LINEAR ALGEBRA
```

Sample Output

```
Refused Javier Jimenez 10
Accepted Arturo America 10
Accepted Carlos Calzada 20
Accepted Carlos Calzada 30
Accepted Carlos Calzada 20
Accepted Felipe Wong 10
Accepted Gonzalo Alvarez 80
Accepted Maria Perez 40
Refused  Alma Garcia 20
```

13th México and Central America
ACM International Collegiate Programming Contest 2008
Problem No. 7 – Egyptian Multiplication
Input file: egyptian.in **Output file:** Standard Output

Ancient Egyptian multiplication is a systematic method for multiplying two numbers that does not require the multiplication table, only the ability to multiply by 2, and to add. Also known as Egyptian multiplication and Peasant multiplication, it decomposes one of the multiplicands into a sum of powers of two and creates a table of doublings of the second multiplicand. This method may be called mediation and duplication, where mediation means halving one number and duplication means doubling the other number.

This method has three phases: the decomposition, the table and the result.

The **decomposition** of a number **N** thus consists of finding the powers of two which make it up. The Egyptians knew empirically that a given power of two would only appear once in a number. For the decomposition, they proceeded methodically; they would initially find the largest power of two less than or equal to the number in question, subtract it out and repeat until nothing remained. (The Egyptians did not make use of the number zero in mathematics).

Example of the decomposition of the number N = 13:

- the largest power of two less than or equal to 13 is 8, 13 – 8 = 5,
- the largest power of two less than or equal to 5 is 4, 5 – 4 = 1,
- the largest power of two less than or equal to 1 is 1, 1 – 1 = 0

N = 13 is thus the sum of the powers of two: 8, 4 and 1.

After the decomposition of the first multiplicand (**N**), it is necessary to construct a **table** of powers of two times the second multiplicand (**M**) from one up to the largest power of two found during the decomposition. In the table, a line is obtained by multiplying the preceding line by two.

For example, if the largest power of two found during the decomposition of **N** = 13 is 8 and **M** = 238, the table is created as follows:

N	**M**
✓1	238
2	476
✓4	952
✓8	1904
13	**3094**

Finally, the **result** is obtained by adding the numbers from the second column for which the corresponding power of two makes up part of the decomposition of **M** (denoted by a mark).

Thus, the result of the multiplication of 13 x 238 is obtained as the addition of: 1904 + 952 + 238 = 3094 **or** 238 + 952 + 1904 = 3094.

Input

The input consists of multiple test cases. Each test case consists of a single line containing two integers **N and M** (**0 ≤ M, N ≤ 1000’000,000**) that indicate the multiplicands and the *begin addition specification*, that it indicates from that row will initiate the addition by obtained the result. The *begin addition specification* is one of **u** or **b** referring to up row and bottom row respectively.

The last test case is followed by -1 on a line by itself.

Output

For each test case, print the case number (beginning with 1) followed by the numbers from the second column (multiplicand **M**) for which the corresponding power of two makes up part of the decomposition of the multiplicand **N** of the form indicated by the *begin addition specification*. Each number (except the first one) is separated by: a space, a character ‘+’, and a space. In the event that the values of N or M are 0, then the answer must be only 0.

Sample Input	Sample Output

`13 238 u`
`13 238 b`
`1000 1 u`
`1 1000 b`
`1 1 u`
`0 10 u`
`-1`
	`Case 1: 13 x 238 = 238 + 952 + 1904`
	`Case 2: 13 x 238 = 1904 + 952 + 238`
	`Case 3: 1000 x 1 = 8 + 32 + 64 + 128 + 256 + 512`
	`Case 4: 1 x 1000 = 512 + 256 + 128 + 64 + 32 + 8`
	`Case 5: 1 x 1 = 1`
	`Case 6: 0 x 10 = 0`