acm High School Programming Contest @

2018 University of Virginia High School Programming Contest

Welcome to the 2018 University of Virginia High School Programming Contest. Before you start the contest, please be aware of the following notes:

Rules

1. There are twelve (12) problems in this packet, using letters A-L. These problems are *loosely* sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Bing Bong to Bong Bong	orange
B	The Cost of Going UP!	dark blue
C	The Next Toy Story	pink
D	A Hard Day at Work	yellow
E	Racer Car Ranking	light green
F	A Brave Archery Tournament	light purple
G	N Queens	gold
H	Communicating with WALL-E	dark green
I	What to Cook?	light blue
J	Finding Nemo	silver
K	Copy That	red
L	Ancestor Age	white

2. Solutions for problems submitted for judging are called runs. Each run will be judged.

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Yes	Your submission has been judged correct.
No - Wrong Answer	Your submission generated output that is not correct or is incomplete.
No - Output Format Error	Your submission's output is not in the correct format or is misspelled.
No - Excessive Output	Your submission generated output in addition to or instead of what is required.
No - Compilation Error	Your submission failed to compile.
No - Run-Time Error	Your submission experienced a run-time error.
No - Time Limit Exceeded	Your submission did not terminate within one minute.

3. A team's score is based on the number of problems they solve and penalty minutes, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty minutes are issued equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty minutes are added for problems

that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty minutes.

4. This problem set contains sample input and output for each problem. However, the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. One challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive a "wrong answer" judgment, you should consider what other datasets you could design to further evaluate your program.

5. In the event that you think a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient; no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still think there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

   You may not submit clarification requests asking for the correct output for inputs that you provide. Sample inputs may be useful in explaining the nature of a perceived ambiguity, e.g., "There is no statement about the desired order of outputs. Given the input: ..., would not both this: ... and this: ... be valid outputs?".

   If a clarification that is issued during the contest applies to all the teams, it will be broadcast to everybody.

6. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first.

   Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams. This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification. This includes submitting dozens of runs within a short time period (say, within a minute or two).

Your Programs

8. All solutions must read from standard input and write to standard output. In C this is `scanf()` / `printf()`, in C++ this is `cin` / `cout`, and in Java this is `System.in` / `System.out`. The judges will ignore all output sent to standard error (`cerr` in C++ or `System.err` in Java). You may wish to use standard error to output debugging information. From your workstation you may test your program with an input file by redirecting input from a file:

```
program < file.in
```

9. All lines of program input and output should end with a newline character (`\n`, `endl`, or `println()`).

10. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

11. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

12. Unless otherwise specified, all numbers in your output should begin with a '-' if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point should be followed by as many decimal digits as can be printed. This means that for floating point values, use standard printing techniques (cout and System.out.println). Unless otherwise noted, the judging will check your programs with 10^{-3} accuracy, so only consider the sample output up until that point.

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure you do not round or use a set precision unless otherwise specified in the problem statement.

13. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no fractional component. Scientific notation will not be used in input sets unless a problem statement explicitly specifies it.

Good luck, and HAVE FUN!!!

A. Bing Bong to Bong Bong

Bing Bong, the pink imaginary friend from Riley's childhood, is lonely. He wants to preserve his memory, but he wants to try to hide his name in the stories. He wants you to help him do this.

Given text input, every time "Bing" is found, replace it with "Bong".

Input Format

The first line of the input file will contain $n \leq 2000$, the number of test cases.

Each test case will start with a single positive integer $c < 100$, followed by c words, all space separated. Note that a word may contain punctuation. All words will be separated by a single space. No test case will consist of more than 1000 characters.

Output Format

For each test case, print the sentence (not the number c from each test case), replacing "Bing" with "Bong". The words in each test case should be space separated, and each test case should be on its own line.

The replacement must be done for three cases: title case ("Bing" to "Bong"), lower case ("bing" to "bong"), and upper case ("BING" to "BONG"). Other cases ("BiNg", "bInG", etc.) should not be replaced. Only whole words should be replaced; do not replace instances of "bing" within other words.

A space after the last word on the line is optional – either way is fine.

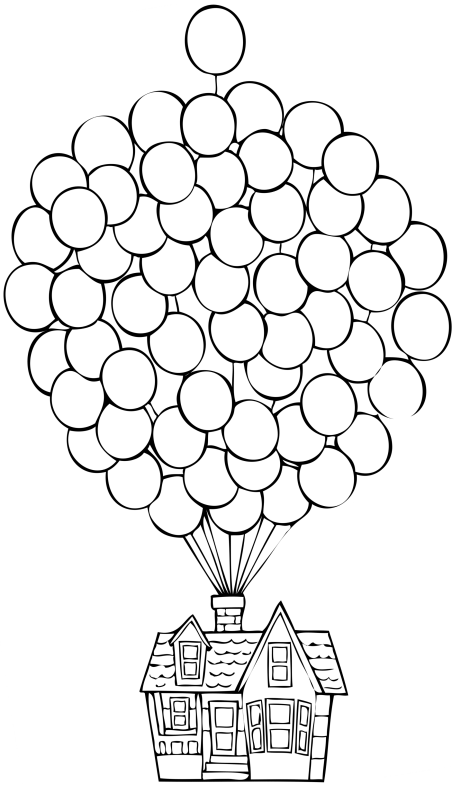
Sample Input

```
5
2 Bing Bong
4 Bing Bong likes Riley
7 Spelled in lower case is bing bong
9 Do not replace the last 4 letters of gabbing
3 No replacements here
```

Sample Output

```
Bong Bong
Bong Bong likes Riley
Spelled in lower case is bong bong
Do not replace the last 4 letters of gabbing
No replacements here
```

B. The Cost of Going UP!



Carl wants to fly his house, using thousands of balloons, to Paradise Falls to fulfill his promise to Ellie. Carl knows exactly how many balloons he needs to get his house airborne. Help Carl figure out how many packs of balloons he needs to buy.

Given the total number of balloons needed, and the number of balloons per pack, determine how many packs Carl needs to buy before he can fly.

Input Format

The first line of input, a positive integer $n \leq 1000$, will be the number of test cases.

The following n lines will each consist of two non-negative integers, separated by a single space. The first will be the number of balloons needed to fly, and the second will be the number of balloons per pack. There will never be zero balloons per pack. Both of these numbers are less than $200,000$.

Output Format

Each case will output one line; that line will contain only the number of packs necessary for that case.

Sample Input

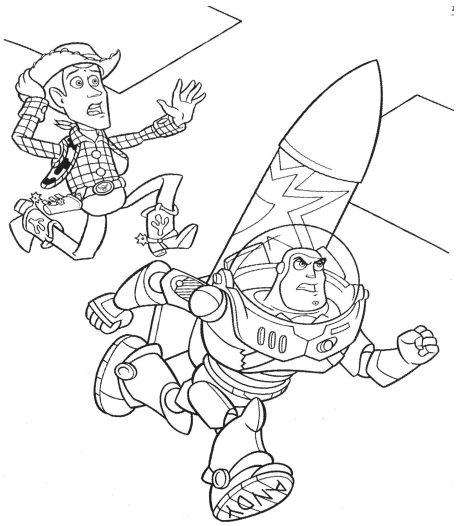
```
3
120097 60
100000 100
81 8
```

Sample Output

```
2002
1000
11
```

C. The Next Toy Story

Toy Story sequels just keep coming out! Fortunately, the Toy Story creators are smarter than other movie producers and know better than to start with, well, movie number 4 and make the movies out of order.



Given a title of a Toy Story movie, find the next sequel.

Input Format

The first line of input will be a single integer, $n \leq 1000$, which is the number of input cases. There will be n lines that follow. Each line is a Toy Story title, consisting of the string "Toy Story" followed by a space, then an integer $-10,000 \leq x \leq 10,000$.

Output Format

For each test case, output the name of the next consecutive Toy Story sequel. The sequel will always have a numerical value one higher than the supplied name.

Sample Input

```
3
Toy Story 5
Toy Story 8
Toy Story -1
```

Sample Output

```
Toy Story 6
Toy Story 9
Toy Story 0
```

D. A Hard Day at Work

Sulley and Mike have a lot of work to do today! They have a number of doors to work on, through which they have to scare children to generate enough power for Monstropolis. They were up late last night, and are rather tired today, so they want to work as efficiently as possible. This means that they want to do the hardest ones done first. Help Sulley and Mike with this problem.

Given a list of doors, and a difficulty rating for each, list them in decreasing order.

Input Format

The input will start with $n \leq 250$, the number of test cases.

Each test case will start with a integer value $1 \leq d \leq 100$, which is the number of doors in that test case. Each door specification will be on its own line. Each door specification will consist of a single positive integer $1 \leq x \leq 1000$ and the door name. A door's name will be a string without whitespace (letters, numbers, and punctuation are valid) and be less than 100 characters long. There will never be two doors in the same test case that have the same difficulty or the same name.

Output Format

For each test case, first output "Case x:" on its own line – note that 'x' starts at 1, and don't forget the colon. Each test case will have the d doors listed in decreasing difficulty order. There should be no extra space between the test cases.

Sample Input

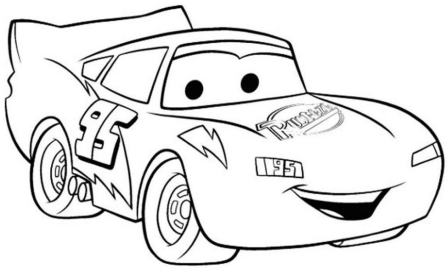
```
2
3
3 Most_Difficult
1 Least_Difficult
2 Moderately_Difficult
8
721 Sophia's_Room
45 Olivia's_Room
554 Emma's_Room
684 Jackson's_Room
461 Aiden's_Room
1000 Liam's_Room
1 Ava's_Room
293 Noah's_Room
```

Sample Output

```
Case 1:
```

```
Least_Difficult
Moderately_Difficult
Most_Difficult
Case 2:
Ava's_Room
Olivia's_Room
Noah's_Room
Aiden's_Room
Emma's_Room
Jackson's_Room
Sophia's_Room
Liam's_Room
```

E. Racer Car Ranking



Lightning McQueen, Strip Weathers, and Chick Hicks are cars that compete in different races against each other. Some races — such as the Piston Cup — are considered more prestigious than others, while others — such as Thunder Hollow — are somewhat less prestigious (we realize only Lightning raced there). Given a list of races of various prestige, and who won each race, help Lightning and his competitors determine who is the best racer.

You will be provided with a list of c cars who are racing in these races. You will also be given a list of r races, each with a name, a prestige, and the first three cars to cross the finish line. Unlike the movie, there are no ties.

The value, v, of a race for the winner is equal to the total prestige of the race. The second place finisher has one-half that value (rounded down), and the third place winner has one-fourth — not one-third! — of the total prestige value (also rounded down). Note that some cars may not finish within the first three placements; these cars receive a value of 0 for such races. Racers accumulate value from each race throughout the entire season.

Input Format

The first line of the file, $n \leq 500$, will be the number of test cases.

The first line of each test case is a string, naming the racing season. The next line contains a single number $3 \leq c \leq 100$, which is the number of cars that are racing. The next c lines contain the name of each competitor as a string. The next line is a number $2 \leq r \leq 100$, which is the number of races to consider. The next r lines will have the information for one race on a single line (space separated), consisting of five values: the name (a string), a prestige value (non-negative integer), and the three winners (first place, second place, and third place, in that order).

All provided strings (racing seasons, car names, and race names) will be strings consisting of letters of either case, numbers, and/or underscores. All integers, as well as all computed values, will fit in a 32-bit signed integer variable.

All finishers in the various races will be listed in the list of c cars, but some cars may never finish in the first three places of a race. Not surprisingly, any given racer can only be listed once for a given race (i.e., one can't place first *and* second place).

Output Format

For each test case, output the racing season name, followed by a colon. The next c lines should contain the names of each racer followed by their total score. The list of racers should be in lexicographical order ("lexicographical" ordering is like alphabetical ordering, but allowing numbers as well). Keep in mind that, in lexicographical order, "car_3" comes *after* "car_20". There should be no blank lines between output cases.

Sample Input

```
1
2005_Season
4
Lightning_McQueen
Strip_Weathers
Chick_Hicks
Slowy_McSlowFace
3
Glen_Ellyn 200 Lightning_McQueen Strip_Weathers Chick_Hicks
Dinoco_400 240 Lightning_McQueen Strip_Weathers Chick_Hicks
Piston_Cup 300 Chick_Hicks Strip_Weathers Lightning_McQueen
```

Sample Output

```
2005_Season:
Chick_Hicks 410
Lightning_McQueen 515
Slowy_McSlowFace 0
Strip_Weathers 370
```

F. A Brave Archery Tournament

Merida wants some help in predicting her score for an archery tournament. When she fires an arrow, we can compute that arrow's trajectory given a quadratic formula and the arrow's initial position and velocity. Then we can compute where on the target it will lie.

An arrow that lands strictly within 1 unit of the center of the target is worth 5 points, if it lands strictly within 2 units it is worth 3 points, and if it lands strictly within 3 units it is worth 1 point. Any further away and the shot is worth 0 points.

We will represent the archery range as a 2D plane. Merida will always shoot directly toward the target, but may miss above or below. The center of the target is always at the point (0,0) and it extends up and down along the y-axis. Given the arrow's initial position and velocity, compute the point value for the shot.

The formulas that govern the arrows' path are:

$x = vt + x_0$

$y = wt - 5t^2 + y_0$

Where t is the time since the arrow was fired in seconds, x and y are the arrow's x and y coordinates at time t, v and w are the x and y components of the arrow's initial velocity, and x_0 and y_0 are the arrow's initial position.

Input Format

The first line of input will be a single integer $n \leq 10,000$; n test cases will follow. Each test case consists of two lines. The first line will be two integers x and y, the x and y coordinates of the arrow's initial position. The second line will be two integers v and w, the x and y components of the arrow's initial velocity as per the formulas above.

Output Format

For each test case, output a single line with a single integer, the number of points for that shot according to the point assignments above.

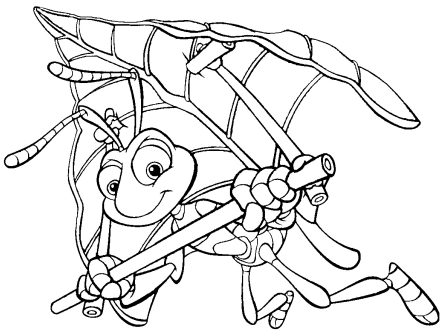
Sample Input

```
2
-10 0
10 5
-10 0
9 4
```

Sample Output

```
5
3
```

G. N Queens



Ant Island has n ant colonies each with 1 queen. Each queen directs 5 teams of $n/2$ size. Each member of those teams has 5 sub-teams of their own of $(n/2)/2 = n/4$ size. Those teams have 5 sub-sub-teams of $((n/2)/2)/2 = n/8$ size (and downward). Teams of size 1 have no sub-teams. Team size must be an integer – you can't have half an ant – so if there are 7 colonies on Ant Island, then each queen will have 5 teams of size 3.

For example, consider the case where $n = 4$. There are 4 colonies, each with 1 queen – that's 4 ants so far. Each of the 4 queens has 5 teams of $4/2 = 2$ ants each. That's $4 \cdot 5 \cdot 2 = 40$ more ants. Each of those 40 ants has 5 sub-teams of $4/2/2 = 1$ ant each – that's 200 more ants. Thus, there are $4 + 40 + 200 = 244$ ants total.

Given n colonies, how many total ants are there on Ant Island?

Input Format

The first line of the input will be a single integer $x \leq 600$, which is the number of input cases.

Each input case will be on its own line, and will contain a single non-negative integer $0 \leq n \leq 500$, which is the number of colonies on Ant Island.

Output Format

Each input case should have a single number printed on its own line. All output values will fit inside a 64-bit signed integer variable (i.e., a `long`).

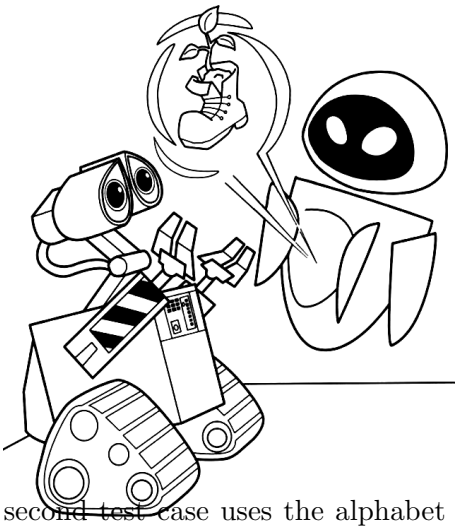
Sample Input

```
4
1
2
3
4
```

Sample Output

```
1
12
18
244
```

H. Communicating with WALL-E



WALL-E and EVE need to communicate, but they are worried that AUTO and GO-4 might be listening in on their conversations. Help them communicate secretly so that they can find the plant and turn the Axiom back toward Earth.

WALL-E and EVE communicate using a substitution cipher. The cipher works by having each letter map to another letter for encryption, and then map back for decryption. Only lower case letters, spaces, and the newline character will be provided as input.

Consider the provided mapping of "defghijklmnopqrstuvwxyzabc", which is the input provided in the first test case below. Since the first letter in that 26 character string is a 'd', that means that 'a' in the plain text will map to 'd' in the cipher text (encrypted version), and 'd' in the cipher text will map to 'a' in the original plain text message. Likewise, 'b' maps to 'e', etc. You will notice that this is a Caesar cipher – but not all the mappings will be such. The second test case uses the alphabet "wjseqvofacugtyhxmpdnikrlzb" – so 'a' maps to 'w', 'b' to 'j', etc.

Given a secret message from EVE that has been encrypted with a substitution cipher, help WALL-E decode it and print the result.

Input Format

The first line of the input will be a single integer $n \leq 1,000$. There will be n test cases that follow.

Each test case consists of two lines: the mapping, and the encrypted text. Recall that each encoded message consisting of lower-case letters (a-z) and spaces. All words will only have once space between them. Messages will not have leading or trailing white space. There will always be at least one word in an input case.

Output Format

For each encoded message, decode it and print it out on its own line.

Sample Input

```
2
defghijklmnopqrstuvwxyzabc
zdooh dqg hyh juhz d sodqw
wjseqvofacugtyhxmpdnikrlzb
winh ad yh ghyoqp ay sfwpoq hv nfq wlaht
```

Sample Output

```
walle and eve grew a plant
auto is no longer in charge of the axiom
```

I. What to Cook?

The renowned restaurant Gusteau's has closed down, and now Remy is working at a new restaurant called Highly Sophisticated Parisian Cuisine (HSPC). However, business is busy, and Remy needs help determining whether he can cook the various dishes that patrons are asking for.

Given a list of ingredients on hand and an ordered list of recipes the guests are asking for, state if a recipe can be cooked with the ingredients left.

If an earlier recipe uses up a given ingredient, then that ingredient is no longer available to successive recipes. For example, assume there are 5 slices of bread in the inventory, the first recipe uses 4 of them, and the second uses 3. The first recipe could be made, but not the second, since there is only 1 slice of bread left over after the first recipe is made. If a recipe can not be made, then there are no ingredients removed for that given recipe. All recipies should be considered in the order provided.

Input Format

The first line of input will be a single integer $n \leq 200$. There will be n test cases that follow.

Each test case will begin with a line with two space-separated integers: $1 \leq i \leq 75$ and $1 \leq r \leq 75$. The i is for how many items are in stock in the inventory, and r represents how many recipies will be specified.

The inventory will be represented by the following i lines; each line will each hold one ingredient. Each line will contain the ingredient count (a positive integer) and the ingredient (an alphanumeric string). Any given ingredient will only be listed once per test case in the inventory.

r recipes will follow. Each recipe begins with a line consisting of an integer $l \leq 10$, the number of ingredients for that recipe. Each of the following l lines will hold one required ingredient for the recipe (and the quantity), represented as they were in the inventory specification. An ingredient not listed in the inventory may be listed in a recipe. Any given ingredient will only be listed once for any given recipe.

Output Format

For each test case, first output the test case number x as "Case x:" (start from 1 and don't forget the colon!) on its own line. Then, iterate through the recipes in that test case; output "Yes" if it can be made with the ingredients remaining or "No" if it cannot. There should be no blank lines between test cases.

Sample Input

```
1
5 3
1 apples
3 onions
2 eggs
```

```
5 beans
1 pepper
3
1 apples
1 onions
4 beans
1
2 beans
1
1 oranges
```

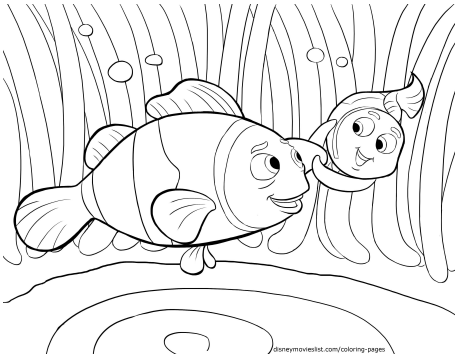
Sample Output

```
Case 1:
Yes
No
No
```

J. Finding Nemo

Can *you* help find Nemo?

Given a word grid, find "nemo" (or similar).



Input Format

The first line of the input file, $n \leq 1,000$, is the number of input cases.

Each input case will have three lines. The first line will contain two space separated positive integers: the width ($2 \leq w \leq 100$) and height ($2 \leq h \leq 100$) of the word grid. The next line will have $w \times h$ letters which are the contents of the word grid. They are in row-major order: meaning the first row is listed, then the second row, etc. All the characters in the word grid, as well as the word to search for, will be only lower case letters.

The last line will contain the word to search for. That word will appear at most once in the word grid provided.

In the sample input given below, all three example cases provide the following as the input (each case has a different word to search for):

```
5 4
dnemoxoxxxxxrxxxxxyx
```

That grid corresponds to the following:

```
d n e m o
x o x x x
x x r x x
x x x y x
```

Output Format

Each input case will have one of two possibilities: either the word is found, or it is not found. Recall that a word will never occur more than once in a given grid.

If the word is found, then the output line will have three space separated values: the x and y coordinates of the start of the word, and the abbreviation for the direction that the word was found. The coordinates start at (0,0) in the upper left corner of the word grid. The direction will be one of: `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, or `nw` (all lower case).

If the word is not found anywhere in the grid, the output for that test case should be "not found".

Sample Input

```
3
```

```
5 4
dnemoxoxxxxxrxxxxxyx
nemo
5 4
dnemoxoxxxxxrxxxxxyx
dory
5 4
dnemoxoxxxxxrxxxxxyx
marlin
```

Sample Output

```
1 0 e
0 0 se
not found
```

K. Copy That

Hello, Mr. Incredible. Yes, we know who you are. Rest assured, your secret is safe with us.

We are infiltrating the Horribly Sinister Plagiarism Corporation (HSPC) to take down their command structure. Unfortunately, they have prepared for such an eventuality by giving their base an annoyingly complex design, which, true to their name, they have copied from another organization. Infiltration of their base will require the talents of someone, well, incredible.

The HSPC command structure is divided into several cells. For each pair of cells A and B, either A controls B or B controls A. But this "control" relation can be cyclic, so it could happen that A controls B and B controls C and C controls A.

We can send in agents to infiltrate any particular cell, which gives us control over that cell and the cells that it controls, but not any other cells. So in the example above, infiltrating A would give us control over A and B, but not C.

For a successful infiltration of the HSPC, we must obtain control over all of its cells. Otherwise, the cells that are out of our control will discover us and start plagiarizing furiously. Our goal is to perform this operation as efficiently as possible. Your mission is to figure out the minimum number of cells we need to infiltrate in order to succeed.

This mission briefing will self-destruct in four hours. Good luck!

Input Format

The first line of the input file will contain $n \leq 100$, the number of test cases.

The first line of each test case will contain the number c of cells the HSPC has ($1 \leq c \leq 75$). Each of the next c lines contains a binary string of length c where the i^{th} character of the j^{th} line is 1 if cell j controls cell i, and 0 otherwise ($1 \leq i, j \leq c$).

The i^{th} character of the i^{th} line is always 0 (no cell "controls" itself). For $i \neq j$, it will never be the case that i controls j *and* also j controls i.

Output Format

For each test case, display "Case x:", where x is its case number (starting from 1, and don't forget the colon), followed by the minimum number m of cells that must be infiltrated to obtain complete control of the HSPC. Then display m numbers, $c_1, \ldots, c_m$, in any order, indicating the list of cells to infiltrate (cells are numbered from 1 to c). If more than one set of m cells gives complete control, any one will be accepted.

Sample Input

```
3
2
00
10
3
010
001
100
5
01000
00011
11001
10100
10010
```

Sample Output

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

L. Ancestor Age

Miguel is trying to determine the ages of his ancestors to make a family tree, and he is given a set of clues that can help him calculate them. Each of his ancestors' spirits gave him a set of arithmetic relationships that reveal the remainders r of their ages when divided by special numbers m. Both r and m are guaranteed to be less than the actual age. Help Miguel determine the ages of each of his ancestors or output "Cannot be determined" if the clues are contradictory. If there are multiple ages that match the criteria, you should output the youngest such age (that is still greater than each of the special numbers). Note that the age of the ancestors and the product of all special numbers will never exceed $2^{31} - 1$.

For example, Miguel found out that if he were to divide the age of his great-great-grandmother by 5, the remainder would be 4. He also knows that if that age is divided by 7, the remainder is 3. Finally, it was revealed to him that if the age is divided by 9, the remainder is 2. From these clues, he was able to conclude that the she is 164 years old. 479 also works with this example, but 164 is smallest solution, so that is the correct answer.

Input Format

The first line of input, a positive integer $n \leq 500$, will be the number of test cases.

The first line of each test case will have a positive integer $e \leq 20$, indicating the number of equations that follow. The next e lines will each have two space-separated numbers, $0 \leq r \leq 100$ and $1 \leq m \leq 100$, where r is the remainder and m is the special number.

Output Format

For each test case, output the minimum number that conforms to the set of given arithmetic relationships. If no such number exists, output "Cannot be determined".

Sample Input

```
4
2
2 3
1 4
2
9 61
4 5
2
20 72
12 16
2
1 6
2 9
```

Sample Output

```
5
314
92
Cannot be determined
```