PROBLEM SET

PREAMBLE

Please note the following very important details relating to input and output:

- Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin` or equivalent. Input will be redirected from a file to form the input to your submission.

- Do NOT prompt for input as this will appear in your output and cause a submission to be judged as wrong.

- Write all output to the screen, i.e. use `stdout`, `System.out`, `cout` or equivalent. Do not write to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen, such as `conio`, `Crt` or anything similar.

- Output from your program is redirected to a file for later checking. Use of direct I/O means that such output is not redirected and hence cannot be checked. This could mean that a correct program is rejected! You have been warned.

- Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by white space, i.e. spaces or tabs.

- An *uppercase letter* is a character in the sequence 'A' to 'Z'. A *lower case letter* is a character in the sequence 'a' to 'z'. A *letter* is either a lower case letter or an upper case letter.

- Unless otherwise stated, a *word* or a *name* is a continuous sequence of letters.

- Unless otherwise stated, a *string* is a continuous sequence of visible characters.

- Unless otherwise stated, words and names will contain no more than 60 characters, and strings will contain no more than 250 characters.

- If it is stated that ‘a line contains no more than *n* characters’, this does not include the character(s) specifying the end of line.

- Input files are often terminated by a ‘sentinel’ line, followed by an end of file marker. This line should not be processed.

Please also note that the filenames of your submitted programs may need to follow a particular naming convention, as specified by the contest organisers at your site.

PROBLEM A **MEMBERSHIP** **3 POINTS**

The Western Suburbs Croquet Club has 2 categories of membership, Senior and Open. They would like your help with an application form that will tell prospective members into which category they will be placed.

To be a senior, a member must be at least 55 years old, and must have a handicap no lower than 8. In this croquet club, handicaps range from -2 up to +26; the better the player, the lower the handicap. In a match, the difference in handicaps is used to determine how many bisques (free shots) the weaker player will receive. A player with no existing handicap will be given a handicap of 26.

Input

Input begins with a line containing a single integer, N (0< N <= 50), which is the number of potential new members for you to classify. This is followed by N lines, each line being data for one member. The data will consist of two integers, A and H, separated by a space. A, the person's age, will be a positive integer, H a valid handicap.

Output

For each line of input, produce one line of output. The output should be the category in which the prospective member would be placed, `Senior` or `Open`.

Sample Input

```
4

18 20

45 2

61 12

58 4
```

Output for Sample Input

```
Open

Open

Senior

Open
```

PROBLEM B **GEOMETRIC PROGRESSIONS** **3 POINTS**

Wikipedia tells us that “In mathematics, a geometric progression, also known as a geometric sequence, is a sequence of numbers where each term after the first is found by multiplying the previous one by a fixed, non-zero number called the common ratio.” So

1 6 36 216

is a geometric progression with a common ratio of 6. As you can see

1 x 6 = 6

6 x 6 = 36

36 x 6 = 216

In this problem you will be given sets of numbers and, if the numbers in a set form a geometric progression, you are to give the common ratio.

Input

Input begins with a line containing a single integer, N (0< N <= 100), which is the number of sets of numbers for you to evaluate. This is followed by N lines, each line being a set of numbers. There will be at least 3 but no more than 20 numbers on a line. If there is a common ratio, its value will be between +1 and +20 or between -1 and -20, both inclusive.

Output

For each line of input, produce one line of output. If the input line is a geometric progression, the output line should be the word “yes” followed by a space, followed by the common ratio. If the input line is not a geometric progression, the output line should be the word “no”.

Sample Input

```
2

1 6 36 216 1296

2 4 8 32 64
```

Output for Sample Input

```
yes 6

no
```

PROBLEM C **COUNTING DIGITS** **3 POINTS**

How many 1s are there in the numbers between 10 and 15 inclusive?

10 11 12 **1**3 **1**4 **1**5

You will see that there are 7. In this problem you will be asked to perform similar counts.

Input

Input begins with a line containing a single integer, N (0< N <= 100), which is the number of counts you have to make. Each count is represented by a separate line containing 3 integers, S F C, separated by single spaces. S (-1000 <= S < 1000) is the start number, F is the finish number (S < F <= 1000) while C is the digit to count (a single digit).

Output

For each count line in the input, produce one line of output. The output should be the number of times the required digit occurs in the specified number range (inclusive).

Sample Input

```
5

10 15 1

1 8 9

-10 10 0

52 160 7

27 398 3
```

Output for Sample Input

```
7

0

3

21

176
```

PROBLEM D **PICTURE ENCODING** **3 POINTS**

Pictures can take up a lot of memory, so ways to encode images to save memory have been invented. In this problem you have to take a fairly simple encoding and turn it into a picture.

The images here are all black and white and have been represented by X characters for black and dot (.) characters for white. The encoding rule is simple. Starting from the left, the number of consecutive white characters is recorded. This will be from 0 to 9 – if there are more than nine, subsequent characters will be encoded separately, preceded by a _ character to show a continuation. The number of consecutive black characters is then similarly recorded.

Your task is to expand the encoded data to produce the original picture.

Input

The input will consist of a number of rectangular images, each represented by encoded data. The first line will contain two integers C and L, separated by a space. C ($0 < C < 100$) represents the number of characters to a line while L ($0 < L < 50$) represents the number of lines in the image. The line 0 0 will mark the end of input and should not be processed.

L lines will follow each consisting of a set of single digits representing the characters in that line. The _ character will be used where more than 9 consecutive characters of the same type occur (see above). Adding the digits will give C.

Output

For each image, output a line of the format `Image n`, where n is the image number. The images are numbered consecutively in the order they appear in the input, starting with 1. The characters on each line of the image must be output with X characters to represent black and dot characters to represent white.

[Turn over for sample input and output]

Sample Input

```
20 22
9_9_2
9_9_2
419_6
329_6
21119_6
519_5
619_4
719_3
819_2
983
9211124
911111114
9211124
911111114
974
91514
91514
91514
91514
91514
9_9_2
9_9_2
0 0
```

Output for Sample Input

```
Image 1
......................
......................
....X.................
...XX.................
..X.X.................
.....X................
......X...............
.......X..............
........X.............
.........XXXXXXXX...
.........XX.X.XX....
.........X.X.X.X....
.........XX.X.XX....
.........X.X.X.X....
.........XXXXXXX....
.........X.....X....
.........X.....X....
.........X.....X....
.........X.....X....
.........X.....X....
......................
......................
```

PROBLEM E **LAB USE** **10 POINTS**

After a number of complaints from pupils about bad behaviour in computer labs, Mr Crossley has introduced some rules with accompanying demerit points. Any pupil who accumulates 50 such points in a week is banned from the lab for 2 days the following week.

Your job is to help Mr Crossley by giving him a list of all pupils in a class who accumulate 50 points in a week.

The following table has been given to all classes:

Unacceptable Behaviour	Code	Demerit points
Bringing food into the lab	FD	10
Using Facebook outside permitted times	FB	25
Using mobile phone in the lab	PH	40
Excessive talking	TK	15
Copying code from another pupil	CP	30
Causing damage to equipment	DM	50
Playing music that can be heard by others	MS	20

Input

Input consists of a number of scenarios, each representing one week. The first line of a scenario consists of a single integer, N, (0 < N <= 40), the number of pupil demerit entries in that week. Input is terminated by a zero value for the number. Do not process that line.

The first line in a scenario will be followed by N lines, with each line containing a name (at most 20 letters with no spaces) and a code as described in the table above. The name and code will be separated by a space.

There may be more than one week's data.

Output

Output will consist of one line for each week. The line starts with the word "Week" separated by a space from the week number followed by the names of the people who are to be banned the following week. Weeks are numbered from 1 in the order they appear in the input. Names must appear in the order that they appear in the input list. There should be a space after the week number and a space between each name. If no pupils were given 50 points in that week, output a message "No pupils banned".

[Turn over for sample input and output]

Sample Input

```
4
Norman TK
Josephine FB
Javier TK
Abdullah FD
8
Shen FD
Shen CP
Norman PH
Shen FD
Maurice FD
Norman MS
Sarah MS
Siobhan PH
0
```

Output for Sample Input	Explanation
`Week 1 No pupils banned`	No pupil has more than 25 demerits.
`Week 2 Shen Norman`	Shen has 50 demerits, Norman has 60.

PROBLEM F **ALPHA PUZZLE** **10 POINTS**

An alpha puzzle is a type of crossword puzzle where each letter square (ie one that is not black) contains a number to represent a letter. Throughout the puzzle, a particular letter is always represented by the same number. All letters of the alphabet are used, so numbers range from 1 to 26.

In this problem you will be given a prepared alpha puzzle and must check it to see that all 26 letters have been used at least once each, making it a valid puzzle. Valid puzzles go to the next stage of preparation, invalid ones go back to the designer.

Input

You will be presented with a number of puzzles to validate. Each puzzle starts with a single integer, S, on a line by itself (10 <= S <=20). S is the size of the puzzle – the number of rows and columns of squares that it contains. If S is 0 it marks the end of input – do not process that puzzle.

There then follow S lines each containing S integers separated by spaces. Integers will be in the range 0 to 26, with 0 representing black squares (ie squares which do not contain letters), while the other integers each represent a letter of the alphabet.

Output

Output one line for each puzzle presented. If the puzzle contains all 26 letters at least once, the output should be “Ready to go”. Otherwise, the output should be “Back to the designer”.

[Turn over for Sample Input and Sample Output]

Sample Input

```
13
1 2 3 1 4 5 6 7 0 0 5 0 7
8 0 9 0 9 0 9 0 7 5 10 11 12
6 9 13 4 11 9 4 12 14 0 11 0 9
2 0 13 0 15 0 4 0 2 0 5 11 15
1 8 15 16 0 17 5 18 15 11 10 0 15
0 0 2 0 17 0 13 0 15 0 0 0 5
19 11 7 2 4 7 0 15 9 6 13 5 6
11 0 0 0 9 0 21 0 12 0 9 0 0
21 0 15 11 13 19 8 7 0 11 15 15 7
13 2 9 0 11 0 9 0 22 0 15 0 2
8 0 23 0 24 9 25 5 8 4 11 13 2
4 2 2 1 16 0 2 0 7 0 2 0 26
2 0 7 0 0 6 4 5 13 2 7 13 7
13
5 7 9 25 20 10 4 0 16 2 16 16 3
21 0 2 0 2 0 13 0 25 0 4 0 7
25 23 23 2 18 0 14 25 26 26 25 11 10
26 0 26 0 12 0 15 0 26 0 5 0 4
26 21 2 13 4 20 10 23 0 15 25 23 20
7 0 0 0 2 0 23 0 9 0 9 0 0
26 19 13 10 25 4 0 1 23 7 10 22 10
0 0 12 0 20 0 25 0 2 0 0 0 8
1 7 4 10 0 5 4 25 16 9 23 25 16
10 0 13 0 17 0 9 0 7 0 25 0 4
6 25 5 5 7 12 10 0 5 25 23 11 2
10 0 24 0 6 0 23 0 25 0 10 0 20
23 21 3 14 10 0 26 16 4 13 23 11 10
0
```

Output for Sample Input	Explanation
Back to the designer	One letter, represented by 20, is missing
Ready to go	

PROBLEM G **FIZZ, BUZZ** **10 POINTS**

Fizz, Buzz is a game for 2 or more players, often children who are learning maths. Each player in turn has to take a number, starting at 1 and going up in 1s. If the number is a multiple of 3, instead of the number they must say "Fizz". If the number is a multiple of 5, instead of the number they must say "Buzz". If the number is a multiple of 3 and of 5, instead of the number they must say "Fizz Buzz". For all other cases, the person says the number. There will be an agreed stopping point. A typical round would start like this:

1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz.

This problem is a variation of that game. You will be given a number of games each with a start number and an end number, and two numbers whose multiples you are to replace by Fizz and Buzz, like 3 and 5 in the above example. You are to display the correct output for each game.

Input

The first line of input will contain a single integer, N ($0 < N <= 20$), which is the number of games to play out. N lines then follow, each representing one game.

A game will be defined by 4 positive integers on a single line, F, L, N1 and N2. F is the first number to be called in the game, L is the last number. $1 <= F <= L < 100$. N1 and N2 are the numbers whose multiples must be replaced. $1 < N1, N2 < 20$.

Output

For each game, the first line must be of the format `Game n`, where n is the game number. The first game in the input list will be Game 1 with games numbered consecutively in the order they are listed.

For each game, all numbers between F and L are to be displayed, each on a separate line. If a number is a multiple of N1, it must be replaced by the word "`Fizz`". If a number is a multiple of N2, it must be replaced by the word "`Buzz`". If a number is a multiple of N1 and N2, it must be replaced by the words "`Fizz Buzz`".

Turn over for sample input and output

Sample Input

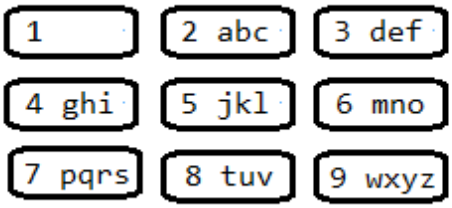
```
1

1 15 3 5
```

Output for Sample Input

```
Game 1
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
Fizz Buzz
```

PROBLEM H **PHONE CYPHER** **10 POINTS**

Tom and Jerry have come up with a simple way of encrypting their text messages to each other. In this problem you will help them by translating encrypted messages into normal text.



Both boys' phones have letters on the keys 2 to 9 as is standard (see diagram). When Tom sends a message to Jerry, he presses the required key one time too many, so gets the letter after the one he wants, unless the letter is the last one on the key when he presses the key just once.

When Jerry sends a message to Tom, he presses the required key one time less than he normally would, so gets the letter before the one he wants, unless the required letter is the first one on the key when he selects the last letter.

The boys do not encrypt digits, punctuation or spaces. All letters are lower case.

Input

Each line in the input begins with a letter to show who sent the message that follows – this will be an upper case J (for Jerry) or an upper case T (for Tom). This is not part of the message, but is there so you know how to decrypt it. The last line starts with # - do not process this line.

Output

For each line of input, output the normal text translation of the message (excluding the initial letter). Each output should be on a line of its own.

Sample Input

```
T xibu bsf zmv emgoh umoghiu?

J sqdscqhmi enq vgd bnmvdrv, ne bntqrd!

#
```

Output for Sample Input

```
what are you doing tonight?

preparing for the contest, of course!
```

Time Limit: 10 sec **Approximation** **NZPCнлмп PROBLEM I**

Often, when doing a programming contest, we need to do some approximate arithmetic – possibly estimating the number of iterations or the memory size required to solve a problem. One fast mental arithmetic technique is known as RUNT (Round-Up Numbers Technique). Arithmetic in RUNT works as follows:

- A number in RUNT is zero or has exactly one non-zero digit.
  0, 100, 5, 5000 are valid RUNT numbers.
  101, 99, 5.5 and 5500 are not valid RUNT numbers.
- When we are given an arithmetic problem we immediately 'round' all numbers to the nearest RUNT number
- After doing an arithmetic operation we 'round' the result to the nearest valid RUNT number.
- RUNT 'rounding' works as follows: if the second significant digit is 2 or less, we round down; if 3 or greater, we round up. For example: 1200 rounds to 1000; 1091 rounds to 1000; 1300 rounds to 2000. There are similar rules for fractions and negative numbers, but they are not required for this problem.

Your task is to write a RUNT desk calculator. Your calculator should accept arithmetic expressions involving non-negative integer literals, parentheses '(' ')', addition '+' and multiplication '*' (no other operators are required). Normal operator precedence applies:

- Parenthesized expressions first.
- Multiplication has precedence over addition.
- In a sequence of additions or multiplications, operations are performed from the left.

Note: Of course such a calculator should never be used. You should think of it as an aid to training, not a useful means of performing calculations.

Input

On the first line of input is a single number N: 1 <= N <= 100, being the number of expressions to evaluate. On the next N lines are N arithmetic expressions. No expression is more than 1000 characters long. There may be space characters before, after or between tokens in an expression. Every expression contains at least one integer literal. All expressions are syntactically valid (parentheses match, literals and operators are used correctly).

Output

One line for each expression, being the integer result of doing the RUNT calculation. Note that all arithmetic can be done using 32 bit integers.

Sample Input

```
6
  2299
100 +   29
135+235
120 + 120+120
4 + 5 * 5
8 * ((3 + 2))
```

Sample Output

```
2000
200
500
300
40
40
```

Time Limit: 60 sec **Glass Cutting** **NZPCнлмп PROBLEM J**

Window glass is manufactured in large rectangular sheets. An important problem in the glass business is trying to cut these sheets into the various pane sizes required by customers as efficiently as possible. PC Glass Limited is working on a new strategy to address the problem and has asked you to develop software to help. Their system is as follows

- Sizes are measured using a new unit – the 'Glass Regular Inch' ('grinch' for short – the grinch is close to an inch, but just a little smaller for easy conversion to millimetres (25 mm/grinch)).
- Glass sheets are manufactured with dimensions that are always an integral number of grinches. For example, they may manufacture 100 by 200 grinch sheets.
- Glass panes are sold in a set number of standard sizes (all rectangles in integral grinch sizes).
- Depending on stock levels and demand, PC Glass chooses and makes regular alterations to the prices at which they sell panes of different sizes.

After setting prices, PC Glass wants software to determine an optimal cutting pattern. The software will be given a sheet size and a list of required pane sizes with prices. It must decide how best to cut one sheet into panes, so as to maximize the value of panes obtained (less the cost of the cuts). The collection of panes that results is of concern. For example, if a sheet is cut entirely into panes of one size and no panes of other sizes are produced and this happens for too long, PC Glass will simply alter the prices to favour other pane sizes and run the program again.

Glass cutting has one unusual feature that must be taken into account. The method of cutting a sheet of glass is to score (scratch) it in a straight line, and then to snap the glass along the scored line. This means that the cut must go all the way from one side of the glass to the other. Further the glass cutting tables at PC Glass only allow cuts parallel to an existing side of a piece of glass – always leading to rectangular panes.

Input

Input will start with a single line holding one positive integer ($0 < N < 100$), being the number of cutting problems to solve. This will be followed by data for each of the N problems. The data for a problem starts with a line of four numbers: W, H, C and S. W and H are the width and height of the glass sheet in grinches: $1 <= W, H <= 500$. C is the cost of making one cut: $0 <= C <= 1000$. S is the number of sizes of pane that may be cut: $1 <= S <= 20$. Next are S lines, each with three numbers: w, h and p. Where w and h are the width and height of a pane and p is its price. $1 <= w <= W$, $1 <= h <= H$. Note that prices are positive integers - all finance is managed in integral dollar amounts.

Output

One line per problem, holding a single number – the maximum price obtainable by cutting the given sheet of glass with the given pane price structure, less the cost of making the cuts. Notes:

- The cutting pattern for maximum price may not be unique.
- The orientation of the glass is not important – 1 by 3 grinch pane may be cut as a 1 by 3 or as a 3 by 1.
- There may be unused glass that is wasted after making cuts

Sample Input

```
4
4 3 5 1
2 1 10
3 4 5 3
3 3 10
1 2 3
1 3 4
3 4 20 3
3 3 10
1 2 3
1 3 4
3 4 1 3
3 3 10
1 2 3
1 3 4
```

Sample Output

```
35
9
0
13
```

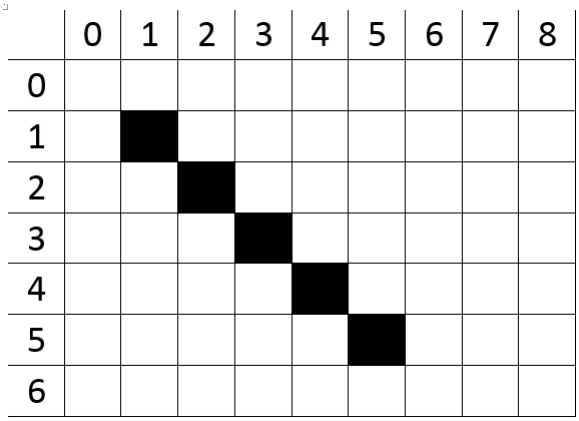
Time Limit: 10 sec **Line Ends** **NZPC2014 PROBLEM K**

In 2D graphics, an important function is line drawing. When lines are thin we don't have to think very much about the way they join. However, if we use wide lines, joints become a concern. In the picture to the left a wide horizontal line has been drawn, connected to a wide vertical line. Each line is many pixels wide, but stops exactly at its end point. The result is an unsightly gap at the joint. In 2D graphics libraries there is usually a mechanism for adding 'caps' to joints. Different geometries are used. Two common forms are the rounded cap and an angular one.

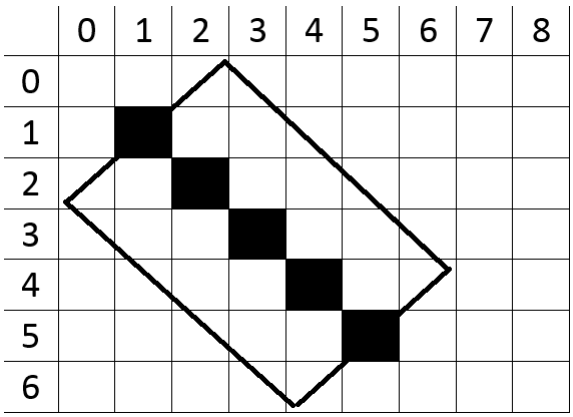
For lines joining at different angles we can use a version of the angular cap made by extending the sides until they meet giving an extended pointed cap. This shape is especially useful in forming arrow.

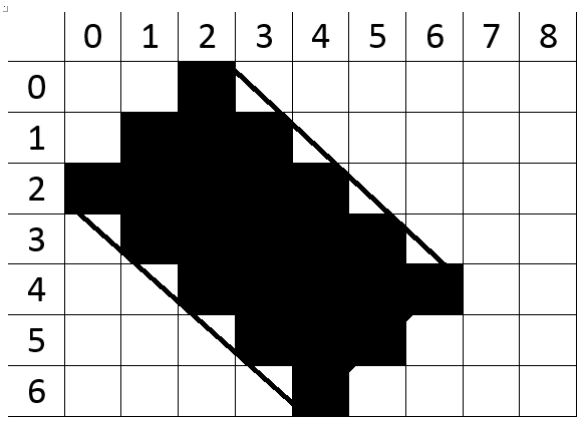
You task is to write a program to accurately determine which pixels to fill in a low resolution 'graphics' environment when drawing and joining two wide lines using the 'angular' cap as above.

Think of your 'graphics' surface is an N * M grid of cells, columns indexed from 0 to N-1 and rows from 0 to M-1.



The diagram shows 9 by 7 grid with a thin 'line' drawn from cell (1, 1) to (5, 5). Coordinates refer to the centre of a referenced cell. Wide lines will have pixels filled to either side of the line. For a line of width W, consider a rectangle of width W around the line, as follows.





Any cell (pixel), whose cell centre lies exactly on or inside the rectangle should be coloured. (Note that precise calculations are required here.)

Finally, because we cannot use a graphics environment in a programming contest, we will represent our graphics surface as an array of characters – '@' representing black and the space character representing white. (This will only look correct with a non-proportional font.

Input

Input will consist of a number P of problems. The first line holds the number P. This is followed by P problem descriptions. Each problem is a sequence of connected line segments to be drawn. The first line for each problem has four non-negative integers $1 <= N, M, W <= 100$ and $1 <= L <= 5$. N and M are the number of columns and rows of the drawing surface, W is the width of the lines to be drawn and L is the number of points to be connected by line segments. Then follow L lines each with an x (column), and y (row) index representing a point on the surface $0 <= x <= N$, $0 <= y <= M$.

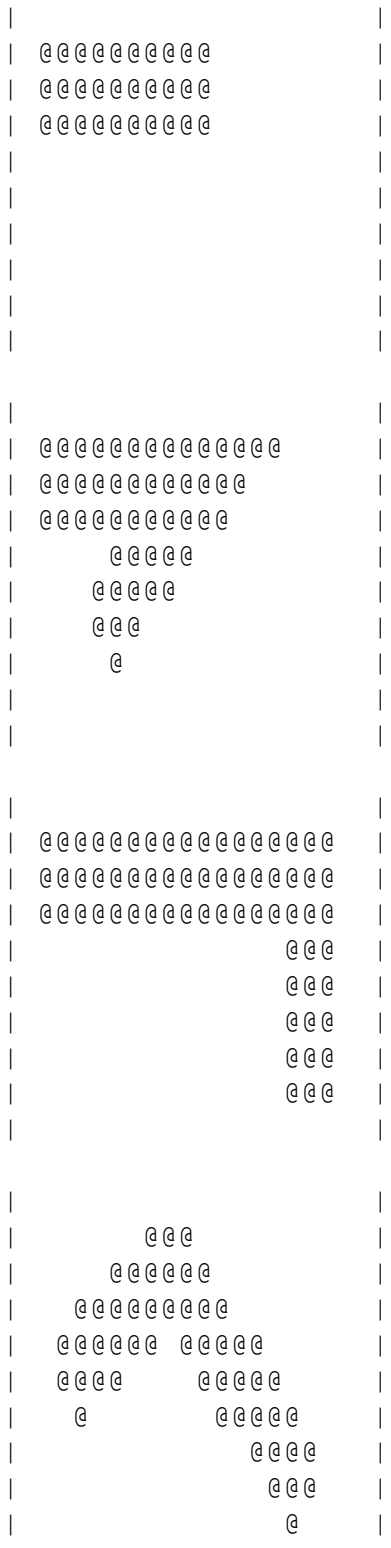
Output

For each problem, a blank line followed by a grid of @ and space characters of the specified size with the requested line segments 'drawn' in @ characters. Each line of the grid should be started and finished with a '|' character. Some lines and caps will extend off the grid – just 'draw' the pixels that are on the grid.

Sample Input

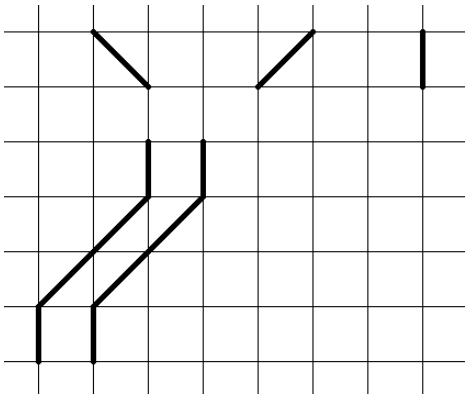
```
4
20 10 3 2
1 2
10 2
20 10 3 3
1 2
10 2
4 6
20 10 3 3
1 2
16 2
16 8
20 10 3 3
2 5
8 2
16 8
```

Sample Output

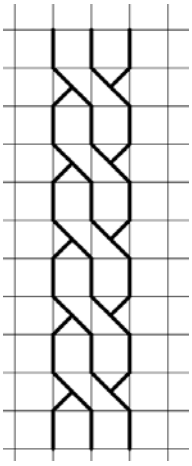


Time Limit: 10 sec **Rope Patterns** **NZPCнлмп PROBLEM M**

A fun children's drawing activity is making patterns on grid paper. By drawing a systematic arrangement of vertical and diagonal strokes in the cells of the grid paper it is possible to produce interesting pictures. For example, with diagonals and vertical lines we can make a wandering pipe.

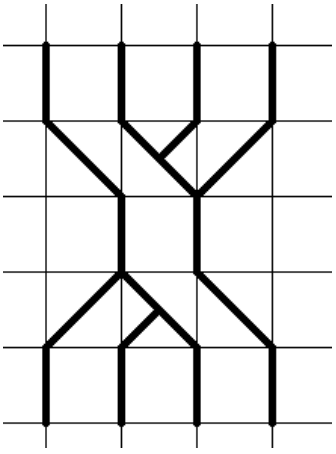


If we also allow half diagonals, it is possible to give the impression that pipes twist around each other. We can think of these new forms as sketches of rope with twisted strands.



You have been approached by the author of a children's activity book to write a program to help them check the accuracy of drawing instructions provided for drawing pipe/rope pictures on a grid. Your task is to take a coded version of a picture, decide if it represents well-formed strands and if so, how many strands there are. A picture is well formed if:

- The top and bottom rows of the diagram have only vertical line segments and empty cells.
- It is possible to interpret the picture as a series of strands of width 1 running all the way from top to bottom. There should be no line segments that are not part of a strand.
- Although a strand can 'pass behind' another strand, at least a part each strand must be visible on every row of the diagram grid. For example, the picture on the right is not well formed even though we can imagine a strand passing from top right to bottom left, because it is not visible on the third row. *Note that the code for this picture is the second to last example in the sample input below.*
- A strand in a well formed picture starts on the first row and extends to the last row of a diagram.



Input

The first line of the input is a number P, being the number of diagrams to analyse. Following are P diagram descriptions. Each description starts with a line holding the number N of grid rows in the diagram and then has N lines of coded grid cell descriptions 1 <= N <= 100. Coding is as follows: A space represents an empty grid cell, the '|' character represents a grid cell with a vertical line down its left edge; a '\' character represents a top left to bottom right diagonal; '/' is the other diagonal. Following a '\' or '/' may be the letter 'u' denoting a top half diagonal of the opposite sense or the letter 'd' denoting a lower half diagonal. The number of cell descriptions on a line is between 0 and 100 inclusive.

Note that there is an asymmetry in the set of patterns that can be drawn as a result of '|' representing a line down the left edge of a cell. This is not desirable and later versions of the coding scheme may fix the problem, but for the program you are writing, the limitation is exists and should be taken into account.

Output

One line for each diagram. If the diagram is not well formed the output should be 'Not well formed'. If the diagram is well formed the output should be 'Well formed with S strands' where S is the largest number of strands that can be interpreted as occurring in the diagram.

Sample Input

```
6
1
|||
3
||||
||||
| ||
3
||||
||||
||||
5
 ||||
//\d\u
|||||
\\/u/d
 ||||
5
 ||||
 \\u/
  ||
 /\d\
 ||||
4
| |
| |
| |
| |
```

Sample Output

```
Well formed with 2 strands
Not well formed
Well formed with 3 strands
Well formed with 3 strands
Not well formed
Not well formed
```

Time Limit: 30 sec **Flash Point** **NZPCнлмп PROBLEM N**

Astronomers have long been interested 'wanderers' – objects in the sky that move against the background of 'fixed' stars – objects like planets, comets and asteroids. The standard method of searching for such objects is: to take a photograph of an area of the sky; wait for a suitable time (days or months); take another photograph of the same area; then compare the photographs. Prior to computers being available, that comparison was manual and time consuming. Now big observatories have software to compare digital images. The NZPC Observatory lacks such software and can't afford to purchase it. Instead they are calling on you to write them a program.

Your program must take information from a pair of processed digitised images and decide if they provide evidence of a wanderer. Of course there are some issues in developing the program. The observatory's telescope alignment system is not terribly reliable, so the images will not be exactly aligned. Observing conditions vary, so the observed intensities of corresponding stars in a pair of images will not be identical.

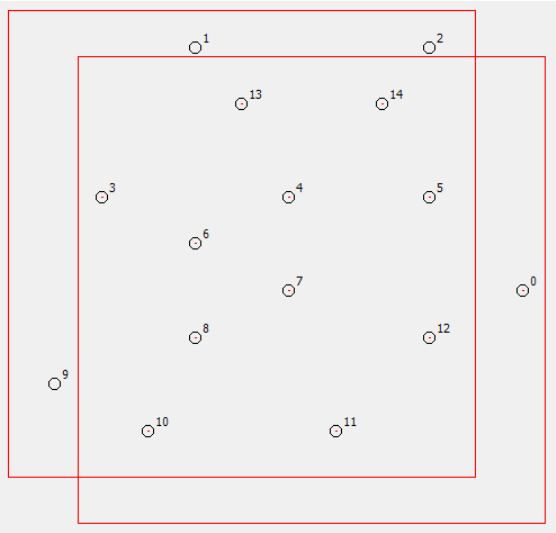
The images used are of small rectangular regions of the sky taken with a digital camera having a resolution of 10,000 by 10,000 pixels. Preliminary processing of each raw image yields a star field array, in which each star is assigned a brightness and reported as occurring at a single pixel location. Brightness is measured on a scale from 1 to 10. Given the problem with telescope positioning, the images in a pair cover only approximately the same area of the sky. It is known however, that the positioning misalignment is never greater than the equivalent of 2000 pixels. Scaling of the images is determined by the geometry of the telescope and is therefore exactly the same. Alignment, in the sense of having parallel X and Y axes respectively is also precise. As a result, the coordinates of stars (pixels) that occur in both images of a pair are offset by some fixed vector (dx, dy), where -2000 <= dx, dy <= 2000. This is slightly complicated by the fact that the real physical (dx, dy) offset of the images is not necessarily an integer. The result is a random pixel alignment effect. In analysing the offsets of corresponding stars across the two images of a pair we will observe one of two values for each of x offset and each y offset (differing by 1). For example images may have stars offset by 100 or 101 pixels in the x dimension. In a similar way the brightness of each star will be consistently different between images with a one intensity unit alignment uncertainty. Note however that the brightness measurement difference may sometimes lead to a star being seen on one image, but being too dim to register on the other.

The professional ridicule that astronomers who make false claims about discovering wanderers suffer is considerable, so the staff have imposed strict conditions on your program. It must only announce that it has found a wanderer if it can account for all the points of light observed in each image with the exception of one point occurring at different locations in the images of a pair, and having the same brightness. Points may be accounted for by: being at corresponding locations with corresponding brightness levels; by being outside the area of view on one image; or being too dim in one image.

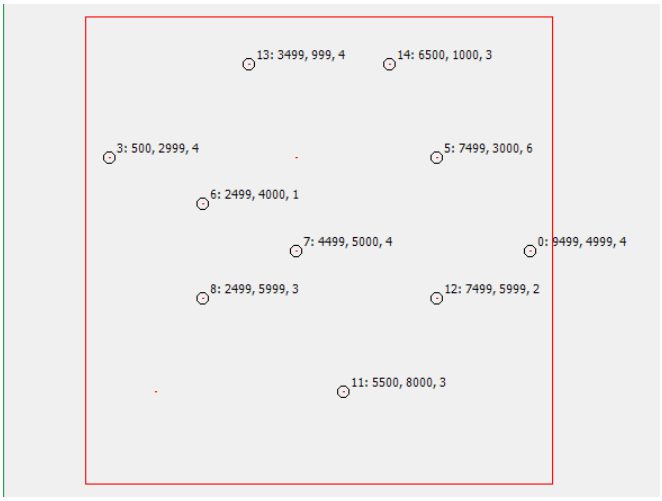
You may assume that at least half the stars in a set will match in location and brightness. You may also assume that the stars are distributed reasonably uniformly and randomly in the images. There will not be more than one location match between images. You may also assume that no two stars occupy adjacent (horizontally, vertically or diagonally) pixel locations (that is a side effect of the operation of the preliminary image processing that has been done).

Example

The picture on the right shows a section of the sky with 15 star locations numbered 0 to 14. Two images are taken as described. The left box is the area captured in the first image and the right box is the area captured in the second image.



The next two pictures show the data captured in each of the images (with the same star location numbers).



Notice that: Star 10 is missing in image 2 because it is too dim. The star indexed as 4 and 8 is actually the wanderer, appearing in different places in the two images. Notice that the x coordinates of fixed stars occurring in both of the two images differ by 1500 or 1501. The y coordinates differ by 1000 or 1001. The brightness values differ by 1 or 2.

Input

Line one holds a single integer N, being the number of image pairs to examine. This is followed by N sets of data on pairs. For each pair the first line of data holds two numbers N1 and N2, the number of stars observed in image one and the number in image two. 10 <= N1, N2 <= 500. Next are N1 lines with star information for the first image and then N2 lines with star information for the second image. Each line of star information contains three positive integers: X, Y and B. X and Y are the coordinates of the star, relative to the top left corner of the respective image. 0 < X, Y < 10000. B is the observed brightness of the point 1 <= B <= 10. The star image information is in random order for each image of the pair.

Output

One line per image pair. The line will either have the text “Couldn’t find wanderer” or 6 numbers: the reported X, Y and B values of the wanderer from image one and from image two of the pair. (It is a requirement of identification that the wanderer is observed in both images.)

Sample Input

```
1
13 10
4000 800 5
9000 800 4
2000 4000 6
6000 4000 7
9000 4000 8
4000 5000 3
6000 6000 6
1000 8000 7
3000 9000 1
7000 9000 4
9000 7000 3
5000 2000 5
8000 2000 4
9499 4999 4
500 2999 4
7499 3000 6
2499 4000 1
4499 5000 4
2499 5999 3
5500 8000 3
7499 5999 2
3499 999 4
6500 1000 3
```

Sample Output

```
6000 4000 7 2499 5999 3
```