1. Almost Bingo

Program Name: Almost.java Input File: almost.dat

Bingo is a game played with 24 integers in the range [1, 75] placed randomly in a 5 x 5 matrix on a card and a free space placed in the middle square of the card. The five columns are named B, I, N, G, and O respectively from left to right. For a card to be valid, it must meet the following requirements:

B column can only have integers 1 through 15
I column can only have integers 16 through 30
N column can only have integers 31 through 45
G column can only have integers 46 through 60
O column can only have integers 61 through 75
There is exactly one free space, it is in 3rd square of the N column, and is marked by FS on the card.

B	I	N	G	O
3	17	33	49	64
6	21	44	56	73
14	25	FS	59	69
9	16	45	46	61
5	30	37	60	70

When Bingo is played, 75 balls are placed in a ball machine and stirred. The balls are ejected from the ball machine one at a time and the Caller calls the letter and number of the ball ejected in the order that they are ejected. A player marks the numbers called on his card as the numbers are called. For a Bingo, the card must have:

- 5 numbers that were called in any row, any column or either diagonal
- The free space may be used to complete any row, column or diagonal in which it falls. For example, B3, I21, FS, G46, and O70 would be a Bingo because they are all on one diagonal.

You are to write a program that checks a bingo card at different points in the game and tells the player the row number, the column letter or which diagonal that he is closest to having a bingo in and how many more numbers he needs to complete the bingo.

- The rows are numbered 1 through 5 beginning with the top row.
- The columns are lettered as explained above.
- The diagonal that goes from top left to bottom right is named `'\'`
- The diagonal that goes from the top right to the bottom left is named `'/'`.

Input

The first line will contain a single integer `n` that indicates the number of Bingo games to follow. For each game, the first five lines will contain the bingo card and the 6th line will contain the letter and the number of the balls in the order that they are called or the letter `x` which marks a check point. This line will not start with an `x`, will not contain consecutive `x` 's, and will always end in an `x`. No balls will be called after a check point results in a valid bingo. All bingo cards and bingo balls will be valid and no bingo balls will be repeated.

Output

Only when an `x` appears in the list of ball calls will you check to see if you have a bingo. If you have a bingo, you will print the line number, column letter or diagonal name of the line or lines in which you have a bingo followed by the word `BINGO`. If you do not have a bingo, you will print the name of the line or lines that have the fewest number of balls that need to be called to complete your bingo followed by the number of balls needed. Print a blank line after the results for each bingo game.

In case of ties, list all ties in the following order: all of the rows in row number order, then all of the columns in order from left to right, then the `\` diagonal and finally the `/` diagonal on a single line and separated by a space.

Example Input File

```
2
1 16 31 46 61
2 17 32 47 62
3 18 FS 48 63
4 19 34 49 64
5 20 35 50 65
B12 I18 G48 x O68 O62 G47 G50 B9 O65 I17 B1 x
1 19 35 47 63
12 21 41 46 73
7 16 FS 59 61
14 29 34 55 70
2 30 40 60 68
I21 B14 I16 B2 B9 N34 I19 N45 G55 x O74 G46 O63 x I29 B1 x
```

Example Output to Screen

```
3 2
\ 1

4 2 I 2 \ 2
/ 1
/ BINGO
```

2. Butterflies

Program Name: Butterflies.java Input File: butterflies.dat

Jane has a database of butterflies that are in the area near where she lives. She needs you to write a program that will search the data base and print a list of all the butterflies that are a particular color.

Input

The first line will contain a single integer `n` that indicates the number of lines in the database followed by the database. Each line of the database will contain the name of one butterfly, a comma, a space and that butterfly's dominant color. The line following the database will contain a single integer `m` that represents the number of lines to follow. Each of the following `m` lines will contain a single color. The names of all butterflies and colors will be uppercase letters.

Output

For each color, print the color followed by a colon. Then, in alphabetical order and on separate lines, print the names of the butterflies in the database that have that color. Print a blank line between the color groups.

Example Input File

```
10
MONARCH, ORANGE
CLOUDED SULPHUR, YELLOW
LYSCIDE SULPHUR, YELLOW
PAINTED LADY, ORANGE
GIANT SWALLOWTAIL, BLACK
CLOUDLESS SULPHUR, YELLOW
BLACK SWALLOWTAIL, BLACK
TIGER SWALLOWTAIL, YELLOW
GULF FRITILLARY, ORANGE
SOUTHERN DOGFACE, YELLOW
2
YELLOW
BLACK
```

Example Output to Screen

```
YELLOW:
CLOUDED SULPHUR
CLOUDLESS SULPHUR
LYSCIDE SULPHUR
SOUTHERN DOGFACE
TIGER SWALLOWTAIL

BLACK:
BLACK SWALLOWTAIL
GIANT SWALLOWTAIL
```

3. Credit Card

Program Name: CreditCard.java Input File: creditcard.dat

A Visa or MasterCard has a 16-digit credit card number. The issuer identifier number is denoted by the first 6 digits. The last digit is known as the check digit and is generated to satisfy the Luhn check. If you take away the 6 identifying digits and the last check digit, then you have the 9 digits that form your account number.

The following algorithm gives the Luhn check:

1. Starting with the leftmost digit, double every second digit (going from left to right). Do **not** double the check digit. In all you will double 8 digits.
2. If the doubling of a digit results in a 2-digit number, add the two digits to form a single digit.
3. Replace the corresponding digits in the original credit card number with the new digits. There will be 8 replacements in alternate positions.
4. Add all the 16 digits in the new set of numbers.
5. If the sum is evenly divisible by 10 then it is a valid credit card number, otherwise it is not.

Let us suppose that the credit card number is 5490 1234 5678 9128, as shown in the first row below with the digits separated.

5	4	9	0	1	2	3	4	5	6	7	8	9	1	2	8
10		18		2		6		10		14		18		4	
1		9		2		6		1		5		9		4	
1	4	9	0	2	2	6	4	1	6	5	8	9	1	4	8

The second row above shows every second digit doubled, starting from the left. The third row shows the doubled digits from row two, with the 2-digit numbers converted to single digits by adding the digits together. The last row shows the number with the changed digits from row three substituted for the original digits from row one.

If you add the last row of 16 digits you get 70 which is evenly divisible by 10, and hence 5490 1234 5678 9128 is a valid credit card number.

Input

The first line will contain a single integer `n` that indicates the number of credit card numbers to test. The next `n` lines will each contain a single 16-digit credit card number (without any spaces).

Output

For each card, print `VALID` if the card is a valid card or print `INVALID` if the card is not a valid card.

Example Input File

```
4
1234567891234567
4552720412345677
9087654321135793
5490123456789128
```

Example Output to Screen

```
INVALID
VALID
INVALID
VALID
```

4. DNA

Program Name: DNA.java Input File: dna.dat

Each strand of DNA is built out of four nucleotides (or bases) called adenine (A), thymine (T), cytosine (C), and guanine (G). Genetic information is determined by the sequence of bases along the strand.

In this problem, you will find the longest common base sequence in two strands of DNA. Each strand is represented by the sequence of letters A, T, C, and G. For example, in the two strands ACTG and TGCA the longest common sequence is TG. The two strands need not have the same length. It is quite possible for the two strands to have no common sequence (a sequence of 1 base does not count) but in no case will the two strands have more than one longest common sequence.

Input

The first line will contain a single integer `n` that indicates the number of pairs of DNA strands that follow. Each pair of DNA strands will occupy two separate lines, one strand per line. There will be `2n` lines to represent all the DNA strands.

Output

For each pair of DNA strands, you will print the length of the longest common sequence followed by a space and then the longest common sequence found. If there is no common sequence your output will be `1`.

Example Input File

```
2
ATGGCATAAGTAT
TGCAGCTGCATCAGGAT
ACTG
TGCA
```

Example Output to Screen

```
4 GCAT
2 TG
```

5. DVDs

Program Name: DVDs.java Input File: dvds.dat

George is very particular (and peculiar) in the manner he stores his DVD collection. He has a series of three crates that were custom-built for his collection. He has named them ALPHA, BETA, and GAMMA. When he watches a DVD during the week, he removes it from the crate, watches it, and then puts it in a pile to be filed in his crates on the following Sunday. When he buys a new DVD, he watches it, and places in the pile with the other DVDs that he has watched and that will be filed on Sunday.

On Sunday, he places the DVDs that he watched during the week into the front of the ALPHA crate in the order he watched them beginning with the DVD he watch first. When ALPHA is full, he moves just enough DVDs from the back of ALPHA to the front of BETA so ALPHA will remain as full as possible. If BETA becomes full, he moves just enough DVDs from the back of BETA to the front of GAMMA so both ALPHA and BETA are always as full as possible. Each Sunday, when his filing is complete, all DVDs will be as far forward as possible.

Each of his three crates can contain at most 6 inches worth of DVD cases. Each case for a single DVD is 0.625 inches thick and each case for a two-disk DVD set is 0.875 inch thick. There will never be more DVDs than the three crates can hold.

George watches at least one DVD each week, and every Sunday he puts them back in the crate in the order he watched them beginning with the one he watched first. For example, consider the week where he watched the DVDs in the following order:

- `Norbit`
- `Speed`
- `My Family: Season 2 (2)`
- `Ghost Rider`
- `King Kong`

Since `Norbit`, `My Family: Season 2 (2)`, and `Ghost Rider` were in the original list of DVDs, they are removed from their positions and all DVDs must be moved forward. Then, the viewed DVDs, including the new one, need to be placed at the front of crate ALPHA. The DVDs, beginning with the DVD at the very front of crate ALPHA will be in the order:

- `King Kong`
- `Ghost Rider`
- `My Family: Season 2 (2)`
- `Speed`
- `Norbit`

followed by `Transformers` and all of the remaining DVDs which have been moved toward the back with the order unchanged as described above.

You are to write a program to keep track of what position (first, second, third, etc.) and in which crate any DVD is located on each Sunday after all of the DVDs have been placed in the crates. Note: There may be some empty space at the end of one crate but not enough empty space to hold another DVD so it will be the first DVD in the next case.

Input

The first line will contain a series of DVD titles that are in the crate at the beginning. The DVD titles will be separated by a comma and a space. DVD titles with a space and the number two (2) in parenthesis at the end of the title are two-disc DVD sets. All other DVDs are single DVDs. Notice that some lines in the actual file, including the first line, may appear on the printed page to be more than one line.

The second line will contain a single integer `n` that indicates the number of weeks the program will be used. Each of the next `n` pairs of lines is considered to be a test case.

- The first line of each test case will have the titles of DVDs, separated by a comma and a space, that are one of two types. Either it is a DVD in the original list that needs to be removed from the crate where it is located so it can be viewed or it is a new DVD. In either case, the DVD will be viewed and then, on Sunday, placed in the front of the ALPHA crate as described above.
- The second line of each test case will have the name of a target DVD that George wishes to locate after all DVDs are placed in the crate.

Output

On a single line for each week, you are to print the name of the crate in which the target DVD is located, a space, and the position of the target DVD within the crate in which it is located.

Example Input File

```
Transformers, Blades of Glory, Disturbia, 300, Spider-Man 3 (2), Ghost Rider,
TMNT, Star Trek Generations (2), My Family: Season 2 (2), Norbit, Because I
Said So, Eragon, Walk the Line, The Holiday, Grounded for Life Season 5 (2)
2
Norbit, Speed, My Family: Season 2 (2), Ghost Rider, King Kong
Eragon
Blades of Glory, Because I Said So, The Holiday, Grounded for Life Season 5
(2), Crash, Speed, Shark Tale, Curious George, Ice Age
King Kong
```

Example Output to Screen

```
BETA 5
BETA 1
```

6. Fibonacci Base

Program Name: FibonacciBase.java **Input File: fibonaccibase.dat**

The Fibonacci series is obtained by starting with 0 and 1, and each subsequent term in the series is obtained by adding the previous two terms. Given `n` in the top row of the table below, the numbers in the second row represent `Fib(n)`, the numbers in the Fibonacci series. So `Fib(0)=0`, and `Fib(1)=1`. Then `Fib(2)` is obtained by adding `Fib(0)` and `Fib(1)`, or `0+1=1`.

n	0	1	2	3	4	5	6	7	8	9	10
Fib(n)	0	1	1	2	3	5	8	13	21	34	55

There are many applications of the Fibonacci series both in mathematics and in the real world. For example, we can represent any positive integer as the sum of one or more of the terms in the Fibonacci series without repetition of any term in the series. For a unique representation, you may not take two successive terms in the Fibonacci series.

Input

The input is an unknown number of lines, each containing a single positive number `n` ($0 < n < 1000$).

Output

For each value of `n`, you will print the Fibonacci terms that add up to it. The Fibonacci terms shall be in descending order and you may not use two successive terms in the Fibonacci series.

Example Input File

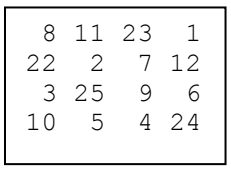
```
16
29
53
92
```

Example Output to Screen

```
16 = 13 + 3
29 = 21 + 8
53 = 34 + 13 + 5 + 1
92 = 89 + 3
```

7. Magic Square

Program Name: Magic.java Input File: magic.dat



At the Math Club's annual banquet, the after-dinner speaker asked one of the students to give him an integer between 34 and 100. The student said 43. The speaker then created a 4 x 4 magic square in which each column, each row, and each diagonal summed to the magic number 43. Additionally, the numbers in the four corners added to 43 as did the "horizontal pairs" (22, 3 and 12, 6), the "vertical pairs" (11, 23 and 5, 4) and the center four numbers (2, 7, 25, 9). You are to write a program to determine the magic number and check to see that all of the criteria have been met.

Input
The first line will contain a single integer `n` that indicates the number of 4 x 4 squares to follow.

Output
If the square meets all of the criteria listed above, print the magic number and `MAGIC`. If it is not a magic square, print `NOT MAGIC`.

Example Input File
```
2
8 11 23 1
22 2 7 12
3 25 9 6
10 5 4 24
8 11 23 1
22 2 7 12
3 25 9 6
10 5 4 19
```

Example Output to Screen
```
43 MAGIC
NOT MAGIC
```

8. Making a Difference

Program Name: Difference.java Input File: difference.dat

Leslie has been struggling in Algebra so her teacher has agreed to give her the left-over tenths of points from the grades of all students in the class whose average is above 90. For example, if John's average is 92.3, Leslie will get .3 points added to her average. If Joe's average is 89.7, Leslie will get no points. You are to write a program that will compute the number of points that Leslie will receive and add that value to her average.

Input
The first line will contain Leslie's average rounded to tenths. Each of the remaining lines will have a single student's average rounded to tenths.

Output
Print Leslie's new average rounded to the nearest integer.

Example Input File
```
67.1
92.1
78.4
62.4
93.4
90.0
99.9
90.8
88.2
```

Example Output to Screen
```
69
```

9. Paint

Program Name: Paint.java **Input File: paint.dat**

You have taken a job as a house painter for the summer. You have decided to write a program to determine how many gallons of paint you will need to complete each job. You will be given a floor plan of the house to be painted and a starting position marked by an asterisk (`*`). The walls of the rooms are marked by a string of `X`'s, open doors are marked by one or more dashes (-), and closed doors are marked by one or more equal (=) signs. You are to paint the walls and ceiling of all rooms that you can enter from the starting position without opening a door.

You have decided to write a program that will mark all areas to be painted in this order:

1. any open door and the spaces on either side of the open door will be marked with a `C` to denote a ceiling area.
2. all remaining spaces that are along a wall or along a closed door will be marked with a W to denote a wall area.
3. all remaining spaces that can be reached will be marked with a `C`.

Your program will also determine the number of gallons of paint you will need for the job. Each `W` will require 0.2 gallons of paint and each `C` will require 0.1 of a gallon of the same paint.

Input

The first line will contain a single integer that indicates the number of jobs that will follow. For each job, the first line will contain a single integer `n` that indicates the number of rows in the house to be painted. Each of the following `n` lines will contain a string with symbols with only spaces, walls, open doors or closed doors. Additionally, the interior of one room will contain an asterisk (`*`) to indicate the starting position. The outside boundaries will contain only wall or closed door symbols.

Output

For each job, you are to print the completed floor plan followed by the number of gallons of paint that you need to buy for the job. Paint can only be bought in one gallon containers. Output one blank line after each job.

Example Input File

```
1
9
XXXXXX===XXXXXXXXXXXXXX
X           X           X
X     *     X           X
X           -           X
X           X           X
XXXXXX===XXXXXXX--XXXXXXXXXX
X           X           -    X
X           X           X    =
XXXXXXXXXXXXXXXXX===XXXXX==XX
```

Example Output to Screen

```
XXXXXX===XXXXXXXXXXXXXX
XWWWWWWWWWWXWWWWWWWWWX
XWCCCCCCCCWXWCCCCCCCCWX
XWCCCCCCCCCCCCCCCCCCWX
XWWWWWWWWWWXWWWWCCWWWX
XXXXXX===XXXXXXXCCXXXXXXXXXX
X           XWWWWCCWWWCCCWWWX
X           XWWWWWWWWWWXWWWW=
XXXXXXXXXXXXXXXXX===XXXXX==XX
18
```

10. Pyramid Scheme

Program Name: Pyramid.java Input File: pyramid.dat

Pete is a planning on constructing office buildings using a pyramidal architecture. The floor plans of the building must be analyzed to see if the elevations of the building will meet Pete's pyramid criteria. The floor plans have the following features:

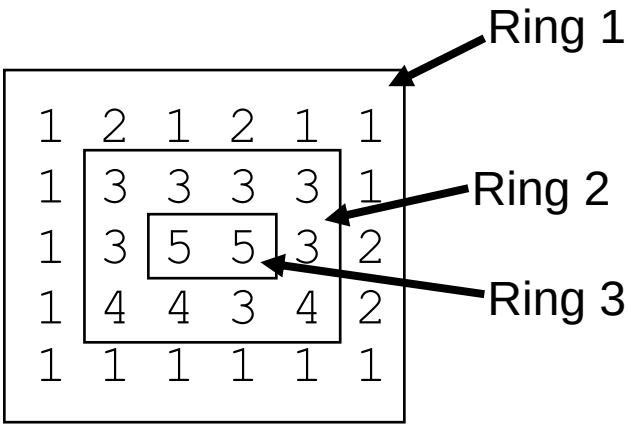
- The building is broken up into one or more concentric rectangular rings.
- Each ring consists of one or more sections.
- Each section has a planned height (elevation), an integer greater than 0.
- Inner rings are completely surrounded by outer rings.
- The overall floor plan of sections and rings is represented as a rectangle of elevations.

Pete's criteria for a correct pyramid do not require every section in a ring be the same height. Instead his criteria require that all of the sections of the outermost ring are lower than all of the sections of the next innermost ring. These criteria must be met by all rings. All sections of a given ring must be higher than all of the sections of the next outermost ring.

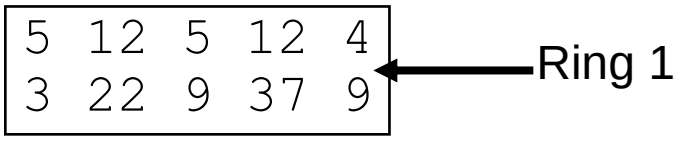
Here is an example of a plan that meets Pete's criteria. Each section's elevation is an integer, separated by a space.

```
1 2 1 2 1 1
1 3 3 3 3 1
1 3 5 5 3 2
1 4 4 3 4 2
1 1 1 1 1 1
```

Here is the same plan with the rings indicated with lines. Notice that every integer in Ring 3 is greater than every integer in Ring 2, and every integer in Ring 2 is greater than every integer in Ring 1.



Here is another example with only one ring.



Input
The first line will contain a single integer `n` that indicates the number of floor plans to be examined. The first line of each floor plan data set will be two integers `r` and `c`, separated by a space. This indicates the number of rows and columns in this floor plan. The next `r` lines of the data set will consist of `c` positive integers each, separated by spaces. These integers indicate the elevations of each section for the proposed floor plan.

Output
The output will consist of `n` lines. For each data set output `BUILD IT` if the floor plan meets Pete's criteria for a pyramid or `SCRAP IT` if the floor plan does not meet Pete's criteria for a pyramid.

Example Input File
```
4
3 4
1 2 3 1
1 2 5 1
1 1 1 1
1 1
12
5 6
11 12 13 14 13 2
13 15 16 16 15 2
11 15 22 22 15 1
11 17 17 15 16 2
11 12 13 14 10 5
2 5
5 12 5 12 4
3 22 9 37 9
```

Example Output to Screen
```
SCRAP IT
BUILD IT
BUILD IT
BUILD IT
```

11. Rose Petals

Program Name: Roses.java Input File: roses.dat

Many people have used pulling petals off roses and tossing them in the air to determine if someone loves them or not. Jane wants to use roses to determine which one of several suitors she will let take her to the prom.

You are to write a program to simulate her decision and determine her prom date. The simulation will work as follows:

- She has a number of roses and she knows how many petals are on each rose.
- The suitors are numbered from 1 to `n`, where `n` is the number of suitors.
- She will select one rose and give one petal to suitor #1 and continuing successively through suitors until she runs out of petals. When she reaches the end of the list, she begins over with the beginning of the list.
- The person who gets the last petal is eliminated from consideration.
- She gets another rose and continues the process beginning with the suitor after the one that was eliminated.
- She continues this process until there is only one suitor left – he is her date for the prom.

Input

The first line will contain a single integer `n` that indicates the number of test cases to be played. For each test case, there are 2 lines. The first line is the list of the first names of the suitors, separated by a comma and a space. The second line has the number of petals on each of the roses separated by a space. The first rose will be used to find the first suitor to be eliminated, the second rose will be used to find the second person to be eliminated, etc. Note: There is one less rose than suitors.

Output

For each test case, you will display the order of the persons eliminated as `LOSER name`, one per line, with the final line naming the winner as `WINNER name` as shown below. Place whitespace between test cases.

Example Input File

```
1
ALAN, BRYAN, CHAD, DUKE, ERIC, FRED
100 234 564 234 1231
```

Example Output to Screen

```
LOSER DUKE
LOSER BRYAN
LOSER ALAN
LOSER FRED
LOSER CHAD
WINNER ERIC
```

12. Time's Up

Program Name: Time.java Input File: none

Time just flies sometimes. It drags when you are at school or work, but when you are free, able to do what ever you choose, time just goes shooting by. How can it be so subjective? The physical constants of the universe don't seem so constant. Class drags on and on. The hands on the clock barely move. But outside things are different. Time rushes by. It flashes past. You feel yourself getting too deep into the philosophical ramifications so you need a nice, concrete problem. Print out the figure below, exactly as shown, to pay your respect to time.

Input
None

Output
Print out the hour glass exactly as shown.

Example Output to Screen
```
++++++++++++++++++++++++++++
-+------------------------+-
--+----------------------+--
---+--------------------+---
----+------------------+----
-----+----------------+-----
------+--------------+------
-------+------------+-------
--------+----------+--------
---------+--------+---------
----------+------+----------
-----------+----+-----------
------------+--+------------
-------------++-------------
------------+--+------------
-----------+----+-----------
----------+------+----------
---------+--------+---------
--------+----------+--------
-------+------------+-------
------+--------------+------
-----+----------------+-----
----+------------------+----
---+--------------------+---
--+----------------------+--
-+------------------------+-
++++++++++++++++++++++++++++
```