1. Attendance: Showing Up Is 80% of Life

Program Name: Attend.java Input File: attend.dat

A school is considering giving awards to students who have good attendance and are on time. They want to look at past attendance records and determine how many awards would be given under various plans. Each plan has a maximum number of days that a student can miss, a maximum number of consecutive days a student can miss, and a maximum number of times a student can be late. Each student receives a single award or no award at all. If a student exceeds one or more of the three limits, that student does not receive an award. If a student does not exceed any of the limits, the student will receive a single award. For each set of past records and each set of rules print out the number of awards that would be given.

Input

- The first line will contain a single integer `n` that indicates the number of plans the school is considering.
- The next `n` lines will each contain 3 integers: `max con late`.
    - `max` indicates the maximum number of days a student can miss school and still get the award, where `max` will be greater than 0.
    - `con` indicates the maximum number of consecutive days a student can miss school and still get the award, where `con` will be greater than 0 and less than or equal to `max`.
    - `late` indicates the maximum number of days a student can be late and still get the award, where `late` will be greater than or equal to 0.
- After the plans will be a single integer `s` that indicates the number of sets of attendance records to test the proposed award plans.
- The first line of each attendance data set will contain two integers: `num size`.
    - `num` indicates the number of students' attendance data in this data set.
    - `size` indicates how many days are in each student's attendance data for this data set.
- The following `num` lines in the data set will each contain `size` characters. All characters will be either `O`, `A`, or `L`, where `O` indicates the student was present and on time, `A` indicates the student was absent, and `L` indicates the student was present, but late.
- The position of the character in a line corresponds to that day. In other words the first character in a line is the attendance status for the first day of school for a particular student, the second character corresponds to the attendance status for the second day of school for that same student, and so forth.

Output

- For each data set print the phrase "`Data Set m`" on a line by itself, where `m` is the number of the data set starting at 1.
- For each of the award plans print the number of awards that would be given under that plan as "`Plan x y`", where
    - `x` is the plan number. Plan numbers, starting at 1, are based on the order they appear in the input file.
    - `y` is the number of awards that would be given under this plan for the given attendance data set.

Example Input File

```
3
3 2 1
4 4 2
1 1 0
2
3 10
OOOOOOOOOO
OOOOAOOLAA
OOAAAOOOOO
4 12
OAOOAOOAOAOO
OOLLOOAAOOOO
AOOLOOOOAOOA
AAAAAAAAAAAA
```

Example Output to Screen

```
Data Set 1
Plan 1 2
Plan 2 3
Plan 3 1
Data Set 2
Plan 1 1
Plan 2 3
Plan 3 0
```

2. Baby Names

Program Name: Baby.java Input File: baby.dat

George is planning on having a lot of children. He dreams of fielding his own basketball, baseball, and football teams with his offspring. The problem is George isn't the sharpest tool in the shed and doesn't have a lot of imagination when it comes to actually giving the kids names. Many times George wants to give a child a name that he has already given to another child. Write a program that assists George in what name to actually give his kids so he can help keep them straight.

George has a *proposed* name for a child when it is born. A proposed name will consist only of upper and lower case letters and internal spaces. If no child has been given the proposed name then that name becomes the child's *actual* name. So for example if George wants to name a child Lisa and he doesn't have any children named Lisa then that child's actual name is Lisa. Now George has another child and he wants to name her Lisa as well. The problem is there is already a child with the actual name Lisa. So the naming algorithm says to give the name Lisa2 to the second child whose proposed name was Lisa. There won't be any children with the proposed name Lisa2 because proposed names can only contain letters and spaces.

Now George has a third child and wants to name her Mary. No child has the name Mary so the child's actual name will be Mary. Then George has a fourth child (remember he is trying to field his own sports teams) and the proposed name is again Lisa. The child's actual name will be Lisa3 because she is the third child to have the proposed name of Lisa.

Names are not case sensitive so Lisa is considered to be the same name as LISA or lISa or other sequences of those letters regardless of case. The actual name will be printed in the same form as the proposed name.

Some names will have internal spaces such as Joe Bob.

Write a program to help George name his children given various scenarios for the number of children born and George's proposed names. Print the actual names George would give the children.

Input

- The first line of input will contain a single integer `n` that indicates the number of data sets to process.
- Each data set will consist of two parts.
    - The first line of a data set will contain a single integer `m` that indicates how many children are in this data set.
    - The next `m` lines will contain the proposed names for the children, one per line. The children are born in the order they appear in the data set. The first line of the data set indicates the proposed name for the first child born, the second line indicates the proposed name for the second child born and so forth.
- Each name will consist of upper and lower case letters (A-Z, a-z) and internal spaces.
- Names from one data set do not affect other data sets.

Output

For each data set print the number of the data set, starting at 1, followed by the actual names given to the children, one name per line. The names are to be printed in the same order as the input, actual name of the first child on the first line, actual name of the second child on the second line and so forth.

Example Input File

```
3
5
Lisa
Lisa
lisa
LISA
lIsA
4
Mike
Lisa
George
Mike
9
Dusty
Lefty
Joe Bob
Dusty
Lefty
Grace
Grace
Bob
Grace
```

Example Output to Screen

```
1
Lisa
Lisa2
lisa3
LISA4
lIsA5
2
Mike
Lisa
George
Mike2
3
Dusty
Lefty
Joe Bob
Dusty2
Lefty2
Grace
Grace2
Bob
Grace3
```

3. Black-out

Program Name: Black.java Input File: black.dat

Black-out is a computer game that is played with a 5x6 grid of "buttons" that are drawn in either black or white. When a player "clicks" on a button, the color of that button is reversed and the color of each adjacent button is also reversed.

For the purpose of this program, the buttons will be displayed as a 5x6 matrix of letters. The letters will be a B for Black or a W for White. A sample Black-out board is displayed to the right.

B	B	B	W	W	W
B	W	B	B	W	W
W	B	B	W	W	B
B	B	W	W	W	B
B	B	W	W	B	W

The object of the game is to Black-out the board, which means to turn all the buttons to B, while abiding by the following rules:

- A button is selected.
- That button and all of its adjacent letters are reversed (B's are changed to W's and W's are changed to B's).
- Buttons are considered to be adjacent only if they touch either horizontally or vertically.

You are to write a program that simulates this game. The rows are numbered from 1 to 5 and the columns are numbered from 1 to 6. Given a list of ordered pairs in the form `row column`, simulate the game and print the state of the game after all ordered pairs have been executed. It is possible for a button to be clicked more than once.

Input

- The first line of input will contain a single integer `n` that indicates the number of games to follow.
- Each game will consist of:
  - 5 lines with six letters on each line, each separated by a space
  - 1 line with an integer `m` denoting the number of ordered pairs to follow.
  - `m` lines, each with an ordered pair in the form `row column`

Output

For each game, print the state of the board after all ordered pairs have been executed. Print exactly one blank line between games.

Example Input File

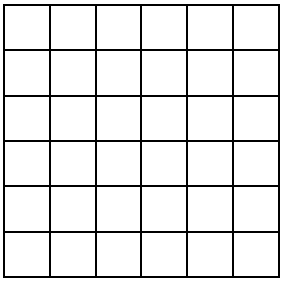
```
1
B B B W W W
B W B B W W
W B B W W B
B B W W W B
B B W W B W
5
1 1
1 4
5 3
2 1
4 5
```

Example Output to Screen

```
B W W B B W
B B B W W W
B B B W B B
B B B B B W
B W B B W W
```

4. Blocks Box

Program Name: Blocks.java Input File: blocks.dat



Rocco has a box for his blocks. The inside measurements of the box are 18" x 18". All of his blocks are 3" wide and have lengths of 3", 6", 9", or 12". If Rocco wanted to completely cover the bottom of the box, he could place thirty-six 3" blocks on the bottom or he could place twelve 9" blocks in the bottom. A diagram of thirty-six 3" spots for blocks is to the right. For the purpose of this program, each 3" spot will be indicated by an asterisk or a period as defined below.

Rocco's mother wants Rocco to learn to problem solve so she places some blocks in place for him. He is to place the other blocks into the box using the following rules:

- Blocks must be placed either vertically or horizontally on the bottom of the box.
- Blocks must lie flat on the bottom of the box.
- He may not move any block that his mother pre-placed.
- No block may overlap another.
- When finished, the bottom of the box must be completely covered with blocks.

You are to write a program that will determine if it is possible for Rocco to cover the bottom of the box given the placement of the blocks by his mother and given the lengths of the blocks he has to use. He does not have to use all of the blocks and he may select the blocks in any order.

Input

- The first line of input will contain a single integer `n` that indicates the number of data sets to follow.
- Each data set will consist of 7 lines.
- Each of the first six lines will contain six consecutive characters consisting only of:
  - An asterisk (`*`) that indicates a 3" spot Rocco's mother covered by a block or
  - A period (`.`) that indicates an empty 3" spot.
- The seventh line will be the lengths, in inches, of the blocks that Rocco can use to cover the bottom of the box, each separated by a space.

Output

For each data set and on a separate line, print `YES` if Rocco can cover the bottom of the box with the blocks given or `NO` if it is impossible.

Example Input File

```
2
......
..*...
..*...
...**.
.**...
...*..
12 6 6 3 9 3 12 9 12 3 6 6
......
..*...
*..*..
.*..**
...*..
..***.
9 12 9 9 3 6 6 6 12 6 12 9
```

Example Output to Screen

```
YES
NO
```

5. Clock Rate

Program Name: Clock.java Input File: clock.dat

A twenty-four hour digital clock displays the time in hours, minutes and seconds. The format of the clock time is `HH:MM:SS`. Leading `0`'s are used to express single digit hours, minutes or seconds with two characters. For example, 3 hours, 7 minutes and 4 seconds past midnight would be expressed as `03:07:04`. The clock time begins with midnight at `00:00:00` and runs for 24 hours ending with `23:59:59`.

Write a program that, given a starting clock time and the amount of elapsed time, prints how many times a particular digit appears. The clock time updates every second. So for example if the start time is `00:00:15` and the elapsed time is 3 seconds, the clock shows the following times:

```
00:00:15
00:00:16
00:00:17
00:00:18
```

The number of times each digit appears given the start time and the elapsed time of 3 seconds is: 0 appears 16 times, 1 appears 4 times, 2 appears 0 times, 3 appears 0 times, 4 appears 0 times, 5 appears 1 time, 6 appears 1 time, 7 appears 1 time, 8 appears 1 time, and 9 appears 0 times .

Input

- The first line will contain a single integer `n` that indicates the number of data sets that follow.
- Each data set will consist of 3 lines.
    - The first line in the data set will be the starting time in the form `HH:MM:SS`
    - The second line will be the elapsed time in the form `<Number of Hours>:MM:SS`. The elapsed time will be greater than or equal to 1 second `(0:00:01)` and less than or equal to 1000 hours (`1000:00:00`).
    - The third line will be a single integer digit `d`, where $0 \leq d \leq 9$.

Output

For each data set, print the number of times the given digit, `d`, appears on the clock, including the start time, during the given elapsed time.

Example Input File

```
5
03:59:59
0:00:01
0
12:34:56
0:00:15
1
23:59:35
0:02:01
5
23:59:59
1:01:01
9
00:00:00
24:00:00
8
```

Example Output to Screen

```
6
20
58
728
24480
```

6. Code Three

Program Name: Code3.java Input File: code3.dat

Sally and Nancy have decided to write to each other in a special code. They decide to use the smallest square matrix that will hold all of the characters in their message. They will place their message, one character per cell, into the matrix in diagonal order beginning with the leftmost column and going diagonally upward and to the right and fill any remaining matrix cells with an asterisk (*). For example, the coded message:

```
DANIOLGA*
```

would have been placed in a 3x3 matrix as:

```
DAN
IOL
GA*
```

so it could be decoded by reading the characters of the matrix in the order (0,0), (1,0), (0,1), (2,0), (1,1), (0,2), and so forth and finally getting the message:

```
DIAGONAL*
```

Input

The first line of input will contain a single integer `n` that indicates the number of coded messages to follow. Each of the following `n` lines will contain a single coded message. Each coded message will contain a perfect square number of characters. There will be no spaces in the input.

Output

For each line of input, print the decoded message with no spaces and with the filler asterisks at the end.

Example Input File

```
3
DANIOLGA*
SlaarlayNadoldydhincicaenmslhdeeoras
SlttooestSvioLthgIa*oUte*
```

Example Output to Screen

```
DIAGONAL*
SallyandNancyaremiddleschoolairheads
SolvethistogotoUILState**
```

7. Hard Money

Program Name: HardMoney.java Input File: hardmoney.dat

Ecco, the puzzler, is trying to figure out how many different ways he can make change for a given amount if he has different denominations of coins. For example current United States coins have denominations worth 1, 5, 10, 25, 50, and 100 pennies. Given these denominations there are various ways to make change for a given amount assuming you have unlimited numbers of coins in each denomination. For example with the current denominations there are 4 ways to have 11 cents. They are 11 pennies, 6 pennies and 1 nickel (worth 5 pennies), 1 penny and 2 nickels, and 1 penny and 1 dime.

But what if the denominations of coins were different? For example, what if the denominations of coins were 1, 3, 11, 17, and 50. In this case there would be 5 ways to have 11 cents. They are 11 pennies, 8 pennies and 1 three cent coin, 5 pennies and 2 three cent coins, 2 pennies and 3 three cent coins, and 1 eleven cent coin.

Write a program that, given the denominations of coins and the goal amount, prints the number of combinations of coins, using the given denominations, that add up to the goal amount. The goal amount and all denominations will be stated in units of pennies.

Input

- The first line will contain a single integer `n` that indicates the number of data sets that follow.
- Each data set will consist of 2 lines.
- The first line in a data set will contain two integers: `g d`.
    - `g` indicates the goal amount, where `g` will be greater than 0 and less than or equal to 10,000.
    - `d` indicates the number of different denominations of coins for this data set.
- The second line in a data set will contain `d` integers in ascending order separated by spaces. These are the denominations of coins for this data set. All denominations will be greater than 0 and less or equal to 1000. No denomination will appear more than once per data set.

Output

For each data set print the number of ways the given denominations can be combined to achieve the goal amount. Assume you have an unlimited number of coins of each denomination.

Example Input File

```
6
11 6
1 5 10 25 50 100
11 5
1 3 11 17 50
100 8
1 2 3 5 7 11 25 50
11 4
2 6 10 25
15 6
2 3 5 7 11 13
100 6
1 5 10 25 50 100
```

Example Output to Screen

```
4
5
98664
0
12
293
```

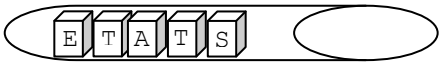
8. I/O

Program Name: IO.java Input File: io.dat

Les has a set of six-sided alpha-dice each of which has the same letter of the alphabet on each side of the die. He has some dice placed on the table in order from left to right and he has two tubes in which he must place the dice one at a time. Each die must be placed in tube #1, removed, and placed in tube #2 in an order so that the resulting "word" in tube #2 spells a specific word.

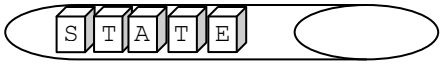
For example:

- Les begins with these letters on the table from left to right in this order: `E T A T S`
- His moves are denoted by:
  - `I` if he is placing the leftmost die from the table into the open end of tube #1.
  - `O` if he is taking the rightmost die out of the open end tube #1 and placing it in the open end of tube #2.
  - `T` if he is taking the rightmost die out of the open end of tube #1 and placing it on the rightmost end of the dice on the table.
  - No dice will be removed from tube #2.
  - There will always be a die available for any move.
- The result of the moves `I I I I I O O O O O` would result in the dice going into tube #1 in the order



tube #1 opening – always faces right

- and then being removed from tube #1 one at a time and placed tube #2 resulting in the dice being in this order.



tube #2 opening – always faces right

Input

- The first line will contain a single integer `n` that indicates the number of line pairs to follow.
- The first line of each pair of lines will contain two words separated by a space.
  - The first word is the letters in the initial order that they appear on the table from left to right .
  - The second word is the target word to appear in tube #2 after all moves have been made.
- The second line of each pair will contain a series of `I`s, `T`s, and `O`s, separated by spaces, that indicate the moves to be made as defined above.

Output

For each pair of lines, print one of the following:

- `NOT ENOUGH MOVES` if, after all moves have been made, there is one or more die still left on the table or in tube #1
- `MATCH` if the table and tube #1 are both empty and the series of moves results in the dice in tube #2 matching the target word.
- `NO MATCH` if the table and tube #1 are both empty and the series of moves does not result in the dice in tube #2 matching the target word.

Example Input File

```
3
ETATS STATE
I I I I O I O O O O
MADER DREAM
I I T I O I I O O O
MADER DREAM
I I T I O I I O O I O O
```

Example Output to Screen

```
NO MATCH
NOT ENOUGH MOVES
MATCH
```

9. Pretty Hands

Program Name: PrettyHands.java **Input File: prettyhands.dat**

Playing cards contain a rank and a suit. The possible ranks are Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, and Ace. The possible suits are Clubs, Diamonds, Hearts, and Spades. A hand consists of one or more cards. A hand is defined to be *pretty* if:

- all the cards in it are Jacks, Queens, Kings, or Aces regardless of suit, *or*
- all of the cards in it have a rank between Two and Ten inclusive and are a red suit. The red suits are Hearts and Diamonds.

Write a program that, given various hands, prints if they are pretty or not pretty.

Input

- The first line of input will contain a single integer `n` that indicates the number of data sets to process.
- Each data set will consist of two parts.
- The first line of a data set will contain a single integer `m` that indicates the number of cards in the data set.
- The next `m` lines will contain the cards in the data set, one card per line.
    - Each card is listed as `<Rank> of <Suit>` where `<Rank>` is replaced with one of the possible card ranks and `<Suit>` is replaced with one of the possible card suits.
    - A given card may appear multiple times in a single data set.

Output

For each data set print `PRETTY` if the cards in the data set constitute a pretty hand per the problem statement or `NOT PRETTY` if they do not.

Example Input File

```
5
1
Ace of Spades
3
Ace of Spades
King of Hearts
King of Hearts
4
Two of Diamonds
Two of Diamonds
Ten of Diamonds
Seven of Hearts
3
Ace of Diamonds
Ten of Diamonds
Seven of Hearts
3
Seven of Spades
Three of Hearts
Nine of Diamonds
```

Example Output to Screen

```
PRETTY
PRETTY
PRETTY
NOT PRETTY
NOT PRETTY
```

10. Star Distances: Computer ... Find the Most Distant Pair ...

Program Name: Stars.java Input File: stars.dat

On board the spaceship USS Kelvin a research team is planning on traveling between the two most distant known stars to perform scientific experiments. Write a program that finds the two stars in the ship's database that are the farthest apart.

Input

- The first line will contain a single integer `n` that indicates the number of stars in the data file.
- The next `n` lines will be the data on stars.
- Each line will consist of the name of a star followed by a colon followed by a space followed by three integers separated by spaces.
- Star names will not have colons in them.
- The three integers are the `(x,y,z)` coordinates for the associated star in a three dimensional coordinate system.
- The distance of each star from the origin `(0,0,0)` and the distance between two stars can be determined by using the following general formula for the distance between two points in a three dimensional coordinate system:

  $$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

- All coordinates will be greater than -100,000 and less than 100,000.

Output

Print the names of the two stars in the data file that are the farthest apart and the distance between them rounded to the nearest tenth. When printing the names of the two stars print the name of the one closest to the origin `(0,0,0)` followed by a comma followed by the name of the second star followed by a colon followed by the distance between them rounded to the nearest tenth. If both stars are the same distance from the origin print the names of the stars in alphabetical order.

Example Input File

```
4
Sol: 1 2 3
Far Far Away: 100 200 -200
Sterupe II: 10 -20 15
Tejat Prior: -10 -5 0
```

Example Output to Screen

```
Sterupe II,Far Far Away:320.5
```

11. Stencil

Program Name: Stencil.java Input File: none

Xavier needs to make a stencil to use to paint his initial on his notebooks. You are to write a program that will create this stencil for him.

Input
There is no input for this problem.

Output
Print this stencil exactly as it appears below.

Example Input File
There is no input for this problem.

Example Output to Screen
```
************************
*  ******************  *
**  ****************  **
***  **************  ***
****  ************  ****
*****  **********  *****
******  ********  ******
*******  ******  *******
********  *****  ********
*********  ***  *********
**********  *  **********
*********  ***  *********
********  *****  ********
*******  *******  *******
******  *********  ******
*****  ***********  *****
****  *************  ****
***  ***************  ***
**  *****************  **
*  ******************  *
************************
```

12. Vowels

Program Name: Vowels.java Input File: vowels.dat

There are many words that contain all of the vowels, A, E, I, O, and U exactly once. There are fewer words that contain all of the vowels exactly once and the vowels appear in alphabetical order.

For example, SEQUOIA is a word that contains all five vowels exactly once. F**A**C**E**T**IOU**S is a word contains all five vowels exactly once and the vowels appear in alphabetical order. **BUTTERFLY** is a word that does not contain all five vowels.

You are to write a program that will determine

- Whether all of the vowels appear in a given word exactly once.
- Whether all of the vowels appear in a given word exactly once and they appear in alphabetical order.
- Which vowels are missing if all of the vowels do not appear in the given word.

Note: All words to be tested will have at most one occurrence of any vowel.

Input

The first line of input will contain a single integer n that indicates the number of words to follow. The next n lines will each contain a single word in all caps.

Output

For each word, print one of the following:

- ALL, IN ORDER if all five vowels appear in alphabetical order
- ALL, NOT IN ORDER if all five vowels appear but are not in alphabetical order, or
- DOES NOT CONTAIN x y … where x, y, etc. are the missing vowels, in alphabetical order and separated by a space, that do not appear in the word.

Example Input File

```
4
SEQUOIA
FACETIOUS
UNIVERSAL
BUTTERFLY
```

Example Output to Screen

```
ALL, NOT IN ORDER
ALL, IN ORDER
DOES NOT CONTAIN O
DOES NOT CONTAIN A I O
```