1. Aliases

Program Name: Aliases.java **Input File: aliases.dat**

Ms. Jones wants to post the grades of her students on the wall but does not want to use any student's actual name for privacy reasons. You are to write a program for her that will create an alias for a given student using Ms. Jones's formula. The alias of a name on her alphabetized roll will be determined by adding the square of its position `p` on the roll, 1 <= `p` <= `n`, to the square of the number of characters in the student's name.

For example, if George is the 3rd name on her roll, his alias would be 45 because $3^2 + 6^2 = 45$

Input

- The first line of input will contain a single integer `n` that indicates the number of lines to follow.
- Each the following `n` lines will contain a list of `m` unique names in alphabetical order and separated by a comma and a single space.
- The first `m-1` names in the list are the names on Ms. Jones' roll.
- The last name in the list is the target name for which you are to find its alias.

Output

For each target name, you will print its alias, each on a separate line.

Example Input File

```
3
Dirk, Evan, George, Harry, Larry, Mo, George
Ann, Beth, Joe Don, Lester, Joe Don
Ben, Carol, Elizabeth, Frank, Gene, Henry, Ted, Zach, Zach
```

Example Output to Screen

```
45
58
80
```

2. All Tied Up

Program Name: All.java **Input File: all.dat**

The only thing more complicated than the U.S. tax code is football tie-breaking procedures. For your junior football league, you've tried to simplify the tie-breakers to determine who wins a division. A division can be thought of as a group of teams. Your league consists of the following teams:

Division A	**Division B**	**Division C**	**Division D**
Armadillos	Bratwursts	Cattle	Donkeys
Anesthesiologists	Banditos	Chefs	Dopplegangers
Apples	Bananas	Coconuts	Donuts

Given a series of games played and the winner of each, write a program to determine between two teams which is better. To determine between two teams which is better:

- The team with the higher Total Win Score is better. The Total Win Score for a team is determined by subtracting its total number of losses from its total number of wins.
- If there is a tie from step 1, the team with the higher Head-to-head Score is better. The Head-to-head Score for a team is determined by subtracting its number of losses against the team it is being compared to from its number of wins against the team it is being compared to.
- If there is still a tie from steps 1-2, the team with the higher Divisional Win Score is better. The Divisional Win Score for a team is determined by subtracting its number of losses against other teams in its division from its total number of wins against other teams in its division.

Input

- The first line will contain a single integer `n` that indicates the number of data sets in the input.
- Each data set will consist of:
  - A line with a single integer `g` that indicates the number of games played, 1 <= `g` <= 10.
  - Then next `g` lines will represent a game played between two teams and will consist of two team names from the above league listing, with a single space separating them. The first team listed is the winner of that game.
  - The final line of the data set will be two team names (again from the above league listing) separated by a single space. These teams are to be compared to determine who is better.

Output

For each data set in the input, output a single line containing the name of the better team of the two given in the final line of the data set. Note that for the purposes of this problem, there will be no ties after applying the above tie-breaker algorithm.

Example Input File (see next page)

2. All Tied Up (cont.)

Example Input File

```
4
4
Armadillos Donkeys
Anesthesiologists Armadillos
Armadillos Donkeys
Cattle Anesthesiologists
Anesthesiologists Armadillos
3
Armadillos Bratwursts
Anesthesiologists Armadillos
Armadillos Apples
Anesthesiologists Armadillos
3
Coconuts Bananas
Bananas Bratwursts
Dopplegangers Donuts
Banditos Bananas
2
Chefs Bananas
Donkeys Donuts
Chefs Donkeys
```

Example Output to Screen

```
Armadillos
Anesthesiologists
Bananas
Donkeys
```

3. Bling

Program Name: Bling.java **Input File: bling.dat**

Nicholas wants to print a diamond of asterisks (*) to use as a pattern for an art project. You are to print his design as shown below.

Input
The input will consist of an unknown number of lines with each line containing a single odd integer `s` where `0 < s <= 35`.

Output
You are to output a diamond of asterisks (*) with `s` rows and `s` asterisks (*) in the longest row as show below. A single blank line should appear between each diamond.

Example Input File

```
25
3
```

Example Output to Screen

```
                    *
                   ***
                  *****
                 *******
                *********
               ***********
              *************
             ***************
            *****************
           *******************
          *********************
         ***********************
        *************************
       ***************************
        *************************
         ***********************
          *********************
           *******************
            *****************
             ***************
              *************
               ***********
                *********
                 *******
                  *****
                   ***
                    *

  *
 ***
  *
```

4. Closet OCD

Program Name: Closet.java **Input File: closet.dat**

The only problem with someone else doing your laundry is that they always hang up clothes in the wrong places in your closet. Now you have to go set things right, but you want to do it with the least amount of possible effort.

Given a sequence of numbers representing articles of clothing on a rack, determine the least number of moves required to return the sequence to ascending order.

A 'move' simply removes a number from one place in the sequence and places it in another. For instance, consider the following sequence of 10 numbers:

9 1 2 3 4 5 6 7 8 10

All that is required is a single move of the '9' to return the sequence to its proper order.

Input
The first line will contain a single integer n that indicates the number of data sets in the input. Each of the following n lines will contain a sequence of the numbers from 1 to 10.

Output
For each data set in the input, output the minimum number of moves required to return the sequence of numbers to ascending order.

Example Input File

```
3
9 1 2 3 4 5 6 7 8 10
10 9 8 7 6 5 4 3 2 1
10 2 3 4 5 6 7 8 9 1
```

Example Output to Screen

```
1
9
2
```

5. Door Man

Program Name: Door.java **Input File: door.dat**

You must escape a burning house, and you know that opening doors can be very dangerous when a building is on fire. Given a map of the house and your starting location, determine the smallest number of doors that must be opened to allow for your escape.

Input
The first line will contain a single integer `n` that indicates the number of data sets in the input. Each data set will consist of:

- a line containing two integers, `r` & `c` (each between 1 and 50, inclusive), indicating the number of rows and columns in this house map
- `r` lines, each containing `c` characters, making up the map. Each character will be one of:
  - `#` - a wall
  - `.` (period) – empty space
  - `d` – a door
  - `S` - your starting location **inside the house**

Note that every house will have at least one door leading to the outside which is reachable from your starting location.

Output
For each data set in the input, output a single line "`X door(s)`", where `X` is the minimum number of doors required to escape the burning building.

Example Input File

```
2
5 5
#####
#...#
#.S.#
#...#
##d##
7 19
###################
#.................d
##.################
#...###............
#.S.d.d............
#...###............
#####..............
```

Example Output to Screen

```
1 door(s)
1 door(s)
```

6. More Boxes

Program Name: More.java **Input File: more.dat**

You've just finished moving into a new apartment and have tons of boxes left over. A friend of yours wants to borrow them and only has access to a compact car to pick them up. So you're trying to pack the boxes into one-another as efficiently as possible.

Given a series of box dimensions, write a program to determine the least possible number of nested boxes you will have to carry to your friend's car. For this problem, a "nested" box is a series of boxes, each nestled inside the next. In the optimal case, you will be able to fit one box inside another until you end up with just 1 nested box. One tactic that is **not** allowed is to put two (or more) boxes "side-by-side" into another. When nesting boxes, you choose exactly one to put inside another, where the dimensions of the outer box are strictly larger than the dimensions of the interior box. Note that it may be necessary to rotate the boxes to make them fit, and you can nest boxes into each other in any order you choose. You are not restricted by the order given in the input.

Input

- The first line will be a single integer `n` that indicates the number of data sets in the input.
- Each data set will consist of
  - A line containing an integer `b` that indicates the number of boxes, 2 <= `b` <= 10.
  - A line containing a space-separated list of `b` box dimensions in the format "`LxWxH`", where `L` represents the length of the box in feet, `W` represents the width of the box in feet, and `H` represents the height of the box in feet, 1 <= `L,W,H` <= 20. Note that although these are the given dimensions, a box can be rotated such that its dimensions change (e.g., a 1x2x3 box can be rotated to be a 1x3x2, 2x1x3, 2x3x1, 3x2x1, or 3x1x2 box).

Output

For each data set in the input, output a single line "`X`", where `X` is the minimum number of nested boxes that can be arrived at given the boxes described.

Example Input File

```
6
2
1x2x3 2x3x4
2
2x3x4 1x2x3
2
2x3x4 2x3x4
2
4x3x2 2x3x4
4
1x2x3 2x3x4 1x2x3 2x3x4
3
2x4x3 9x9x9 1x2x3
```

Example Output to Screen

```
1
1
2
2
2
1
```

7. Number Systems

Program Name: Numbers.java **Input File: numbers.dat**

Different number systems use different symbols to create numbers and order their numbers based on the place value of the different symbols. The number systems for this problem will use a subset of the digits normally used in base ten, the digits 0 through 9, and the value of each digit is the same as in base ten. Each number in the number system must contain each digit in the number system at least once and leading zeros are not used. Some examples follow:

Number System	Digits Used	First Five Numbers
THRONE	1 3	13 31 113 131 133
TWIXEZ	0 2 6	206 260 602 620 2006
FOURNION	1 4 9	149 194 419 491 914

Input
The first line of input will contain a single integer `n` that indicates the number of number systems to be examined. Each the following `n` lines will contain `d` unique digits followed by an ordinal number that indicates which number in the number system that you are to find. Each of the digits and the ordinal will be separated by a space.

Note: There will be no numbers in a number system larger than 100000.

Output
For each number system, you will print the desired number on a single line.

Example Input File
```
4
0 2 6 7
1 7 4 8
3 4 8
7 6 7
```

Example Output to Screen
```
2060
1174
443
767
```

8. Quilt

Program Name: Quilt.java Input File: quilt.dat

Aunt Mabel likes to make quilts but needs some help with her patterns. The squares on the quilt are designed in different colors that, when the squares are sewn together, make a design. Aunt Mabel wants to compare different patterns to her master pattern to see that the new designs match her master pattern with only different colors. Aunt Mabel's patterns are designed in a square of 10 colors by 10 colors.

Aunt Mabel's "master pattern", where each different letter stands for a different color:

```
RRRRRRRRRR
RRRRBBRRRR
RRRRBBRRRR
RRYYYYYYRR
RRRRLLRRRR
RRYYYYYYRR
RRRRXARRRR
RRSSDDSSRR
RRUUUUUURR
RRRRRRRRRR
```

This is a "test pattern" that Aunt Mabel needs to have checked to see if it is the same pattern with different colors:

```
ZZZZZZZZZZ
ZZZZBBZZZZ
ZZZZBBZZZZ
ZZRRRRRRZZ
ZZZZLLZZZZ
ZZRRRRRRZZ
ZZZZXCZZZZ
ZZSSDDSSZZ
ZZOOOOOOZZ
ZZZZZZZZZZ
```

These two patterns are considered to be the "same" because for every color in the master pattern, there is a corresponding color in the same position in the test pattern. For example, for each `R` in the master pattern, there is a corresponding `Z` in the same position (row and column) in the test pattern and there are no other `Z`'s in the test pattern. Similarly, all of the `B`'s in the master pattern correspond to all of the `R`'s in the test pattern, and so forth. If there is at least one character in the test pattern that does not match the master pattern, the test pattern is not the same as the master pattern.

You are to write a program that will check a series of test patterns to see if they are the "same" as the given master pattern.

Input

- The first line of input will contain a single integer `m` that indicates the number of master patterns to follow.
- For each master pattern `m`:
  - The first 10 lines of input will each contain 10 consecutive upper-case letters of the alphabet, representing the master pattern itself.
  - The next line will contain a single integer `n` representing the number of 10 x 10 test patterns to follow.
  - Each test pattern will contain 10 lines with 10 consecutive upper-case letters of the alphabet on each line.

Output

For each test pattern, you will print "`SAME`" if the test pattern is the "same" as the master pattern or "`NOT SAME`" if they are not the "same".

8. Quilt (cont.)

Example Input File

```
2
RRRRRRRRRR
RRRRBBRRRR
RRRRBBRRRR
RRYYYYYYRR
RRRRLLRRRR
RRYYYYYYRR
RRRRXARRRR
RRSSDDSSRR
RRUUUUUURR
RRRRRRRRRR
2
ZZZZZZZZZZ
ZZZZBBZZZZ
ZZZZBBZZZZ
ZZRRRRRRZZ
ZZZZLLZZZZ
ZZRRRRRRZZ
ZZZZXCZZZZ
ZZSSDDSSZZ
ZZOOOOOOZZ
ZZZZZZZZZZ
AAAAAAAAAA
AAAABBAAAA
AAAABBAAAA
AAQQQQQQAA
AAAALLAAAA
AAQQQQQQAA
AAAAZCAAAA
AASSDDSSAA
AAOOOOOOAA
AAAAAAAAAA
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
1
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXOXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
```

Example Output to Screen (see next page)

8. Quilt (cont.)

Example Output to Screen

```
SAME
SAME
NOT SAME
```

9. Short Walk on El Camino Big Num

Program Name: Shortwalk.java **Input File: shortwalk.dat**

Juan and Frank are planning on moving, but they are good buddies and so they want to buy two houses that are close together. They have searched realtor.com and found houses for sale on various streets. They want to live together on the same street and so they want to know how close the closest houses are on a given street. Write a program to determine the difference between the closest addresses on a given street. There may be more than one pair of houses that are closest together on a street based on address so print out the number of pairs of houses that are the smallest distance apart for each street.

Input

- The first line will contain a single integer `n` that indicates the number of data sets that follow.
- For each data set:
  - The first line of a data set will be the name of the street for that data set.
  - The second line of the data set will be a single integer `m` that indicates how many addresses follow, where `m` will be greater than 1.
  - The next `m` lines will be integers representing street addresses, 0 < address < 100,000,000,000,000 (These are very long streets.)
  - No address will appear more than once per data set.

Output

For each data set print out the name of the street, the smallest difference between street addresses on that street, and the number of pairs of addresses on that street that are the minimum distance apart as shown in the Example Output to Screen below.

Example Input File

```
3
El Camino Real
5
1000
750
10000
2000
1
El Camino Big Num
4
9000000000
10000000000
10000000009
1
Sweet Cherry Lane
5
1000
900
1100
9000
9100
```

Example Output to Screen (see next page)

9. Short Walk on El Camino Big Num (cont.)

Example Output to Screen

```
El Camino Real
min distance: 250
number of pairs: 1
El Camino Big Num
min distance: 9
number of pairs: 1
Sweet Cherry Lane
min distance: 100
number of pairs: 3
```

10. Snooze

Program Name: Snooze.java Input File: snooze.dat

When hitting the snooze button in the morning, it's important to know how much you're putting off the inevitable.

Each time you hit the snooze button on your alarm clock, it will wait 9 more minutes before activating again. Given the timing of the initial alarm and the number of times snooze is hit, show what your LED clock display will look like when you finally get up.

Your alarm clock displays digits in the following way using underscore ('_') and pipe ('|') characters:

```
 _       _   _       _   _   _   _   _
| |   |  _|  _| |_| |_  |_      | |_| |_|
|_|   | |_   _|   |  _| |_|     | |_|  _|
```

It uses asterisks ('*') to separate hours from minutes, as in the following example:

```
      _       _
|   _| *   _| |_|
| |_  *   _|   |
```

Input

- The first line of input will contain a single integer `n` that indicates the number of scenarios you need to process.
- Each scenario `n` will have two parts:
  - There are three lines that make up the time in digits that you see on your clock when the alarm first activates.
    - All digits take up a 3x3 array of characters in the input and are right-justified.
    - Digits are separated from each other and the asterisks by a single column of spaces.
    - There is no leading zero for single-digit hours.
  - The fourth line will contain a single integer `m` that indicates the number of times you hit snooze, 0 <= `m` <= 100.

Output

For each scenario in the input, output three lines showing your alarm clock's state when you finally get up.

Example Input File

```
2

  | *   _| |_|
  | *   _|   |
1

    _       _
  |   _| *   _| |_|
  | |_  *   _|   |
3
```

Example Output to Screen

```
      _
  | * |_|   _|
  | *   |   _|

    _
  | * | |     |
  | * |_|     |
```

11. Walk the Line

Program Name: Walkline.java **Input File: walkline.dat**

Given four distinct points on a coordinate plane determine the number of unique lines that pass through the points.

The answer will be either 1, 4 or 6.

- The answer is 1 if all 4 points are collinear.
- The answer is 4 if 3 of the points are collinear and the 4th is not, in which case the 4th forms 3 more lines with the 3 collinear points.
- The answer is 6 if no more than 3 points are collinear. All 6 pairs of points form a unique line.

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 a single line:
  - Each line will consist of 8 integers. These are the 4 pairs of points for the data set.
  - Each pair of integers represent the x and y coordinate of one point in the data set.
  - All points in data set will be distinct.
  - All coordinates of points will be greater than or equal to -1000 and less than or equal to 1000

Output

Print out the number of lines that pass through the points in the data set.

Example Input File

```
3
5 5 6 6 -3 -3 7 7
3 0 1 1 -5 0 2 0
0 0 5 5 0 5 5 0
```

Example Output to Screen

```
1
4
6
```

12. Yahtzee

Program Name: Yahtzee.java Input File: yahtzee.dat

Yahtzee is a game that is played with 5 standard six-sided dice with numbers one through six. While playing the game of Yahtzee, the person who rolls a "Yahtzee" has great possibility of winning the game. A Yahtzee has been rolled when all five dice have the same number. The rules for playing Yahtzee are as follows:

- Each player rolls his dice once.
- Each player may then roll zero or more of his dice a second time.
- Each player may then roll zero or more of his dice a third time.
- After the third roll, the players compare their five dice to see who has the best "hand".

For the purpose of this program, you will be given the outcome of the final roll of each of the three players. You are to determine if any of the players has a Yahtzee and, if more than one player has a Yahtzee, which player or players have the highest Yahtzee. A hand consisting of five ones is the lowest Yahtzee and a hand consisting of five sixes is the highest Yahtzee.

Input

- The first line of input will contain a single integer `n` that indicates the number of games to be played.
- For each game:
  - There will be three lines, one for each player in the game.
  - Each line will have a person's first name followed by the numbers on each of the five dice the player had after the third roll.
  - There will be a single space following the player's name and the numbers on the dice will each be separated by a space.

Output

For each game:

- If there is exactly one person with a Yahtzee, you will print the name of that person.
- If there is more than one person with a Yahtzee, you will print the name of the person with the highest Yahtzee. For example, a roll of 5 5 5 5 5 would beat a person with a roll of 4 4 4 4 4.
- If there are two or more people with the same roll, then there is a tie and you will print, on a single line, the word "`TIE`" followed by the names of the players who were tied in alphabetical order and separated by a space.
- If no hands contain Yahtzee, you will print "`NO WINNER`".

Example Input File

```
3
MARY 5 5 5 5 5
JOAN 4 4 4 4 4
ART 3 3 3 3 3
RON 1 1 1 1 1
ALEX 1 1 1 1 1
CHEZ 1 1 1 1 2
AMY 6 6 3 6 6
REX 6 4 6 6 6
RICK 6 2 6 6 6
```

Example Output to Screen

```
MARY
TIE ALEX RON
NO WINNER
```