University Interscholastic League

Computer Science Competition

2015 District 1 Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Table of Contents

1. Base Fibonacci

Program Name: Base.java Input File: base.dat

The Fibonacci series is obtained by starting with 1 and 2 and subsequent terms in the series are obtained by adding the previous two terms.

n	0	1	2	3	4	5	6	7	8	9
Fib(n)	1	2	3	5	8	13	21	34	55	89

In the above table the Fibonacci series is represented by the row Fib(n). 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 the terms in the Fibonacci series without repetition. For a unique representation, you may not take two successive terms in the Fibonacci series.

Input
The input is a sequence of lines, each having 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 should be in descending order and you may not use two successive terms in the Fibonacci series. If the value of N is a Fibonacci number there is no solution with 2 or more non-successive terms so just print N = N where N is the Fibonacci number.
17 = 13 + 3 + 1 (correct)
17 = 8 + 5 + 3 + 1 (incorrect)

Example Input File
```
16
29
53
55
```

Example Output to Screen
```
16 = 13 + 3
29 = 21 + 8
53 = 34 + 13 + 5 + 1
55 = 55
```

2. Books

Program Name: Books.java Input File: books.dat

Sam is a studious student who wants to buy all the textbooks for all of his classes in one trip to the bookstore. He just has one problem. He can only carry a maximum weight K on a given trip. Given N books and their weights, help Sam figure out the maximum number of books N he can bring back home on his first trip to the bookstore.

Input

The first integer C denotes the number of test cases to follow. For the following C test cases, each case starts with a floating point number K that represents the maximum weight that Sam can carry on one trip and an integer N that is the number of books that are on Sam's list of books to buy ($0 < N < 50$). The next N floating point numbers represents each book's weight.

Output

Print out the maximum number of books that Sam can bring back home on his first trip.

Example Input File

```
3
26.5 5
3.72
6.83
12.684
6.712
9.152
105.293 5
12.34
53.91
6.29
19.6
13.153
85.73 12
22.35
12.43
7.64
10.75
26.587
13.276
18.2
19.325
8.64
17.272
11.933
22.89
```

Example Output to Screen

```
4
5
7
```

3. Collation

Program Name: Collation.java Input File: collation.dat

In preparation for teaching her new kindergarten class, Agnes has made some devious plans. Rather than teaching the children the alphabet in the standard alphabetical order, she's decided to teach the kylographical order. For each test case you will print the words sorted lexicographically using kylographical order.

Input
The first line of input contains T, the number of test cases.

Each test case contains three lines. The first line in each test case is the kylographical order of the alphabet. The next line contains a single integer, *N*, the number of words to be sorted. The last line contains *N* space-delimited lowercase words.

Output
For each test case, print the words sorted lexicographically using kylographical order.

Constraints
```
1 <= T <= 10
1 <= N <= 20
```

Example Input File
```
3
ojhdfnexkizuyvgltraqpwbsmc
5
dance safe sidewalk orange safety
zyxwvutsrqponmlkjihgfedcba
7
cage hammer tree cow yearning treatment morning
fhgdsakjlpioytuerqwczxbnvm
3
apple justice favorite
```

Example Output to Screen
```
orange dance sidewalk safe safety
yearning tree treatment morning hammer cow cage
favorite apple justice
```

Explanation of Output
The letter *o* appears first in the first given kylographical order, so *orange* comes before all the other words. Since the letter *d* appears before the letter *s*, *dance* is the next word. *sidewalk*, *safe*, and *safety*, all begin with an *s*, so they are compared with the next letter. Since *i* appears kylographically before *a*, *sidewalk* is next. Lastly, since *safe* is a sub-word of *safety*, and therefore shorter in length but equal up to its length, *safe* comes before *safety*. In the second kylographical order (the standard alphabet backwards) the output places the words in reverse standard alphabetical order.

4. Face

Program Name: Face.java Input File: none

Long before the advent of sophisticated graphics packages, there were artists who used just the symbols available on the keyboard to make their drawings. Some of their drawings were quite realistic and detailed. They always used mono-spaced fonts like Courier to draw their symbols. Since these fonts were fixed in width, it was easy to align them. Moreover, they used the Space key instead of the Tab key to denote spaces and the vertical bar (|) instead of the capital I to draw vertical lines. In this program you will have a chance to create an ASCII picture.

Input
There is no input for this problem.

Output
You are to output the face as shown below. Each character is of equal size (including spaces). Your output may look wrong and still be correct (it requires a fixed-width font to look the same as below), so make sure you are careful to follow the design exactly. You may assume that there are no trailing spaces on any given line, and no trailing newline. The picture begins with a blank line on the top. There is no blank line at the bottom.

Output to screen

```
      8888888888
    88          88
   88            88
 88                88
 88  ----+  +----  88
888   +-+ || +-+   888
888  + 0 |||| 0 +  888
 88  +   +||+   +  88
 88      (..)      88
 88       ||       88
  88    (----)    88
    88    --    88
     88        88
       88888888
```

5. Festivals

Program Name: Festivals.java Input File: festivals.dat

For the Celtic tribe of Galimatias, the leap year played a major role in determining their festivals. They knew from the Romans that a year is a leap year provided it is divisible by 4 and not by 100 and all years divisible by 400 are leap years. They had two important festivals. The Brimborion festival occurred every year that was divisible by 18 provided it was a leap year. The Narishkeit festival occurred every year that was divisible by 25 provided it was not a leap year.

In this problem, given a year you will have to state what properties that year has. The year could be a leap year and / or festival year. If the year is neither a leap year nor a festival year, then it is an ordinary year.

Input
The first line will contain an integer n, indicating the number of years to follow. It will be followed by n years, where n will be in the range 2 through 10 inclusive. Each year will be on a separate line. All the years will be in the range 500 to 1500 inclusive.

Output
For each input year, output the different properties for that year. Each property should be on a line by itself. The order of printing the properties (if present) is leap year, then Brimborion festival year or Narishkeit festival year or an ordinary year. There are four different properties for the years. A blank line should separate the output for each line of input.

Sample Input
```
4
1017
1400
1200
1080
```

Sample Output
```
1017 is an ordinary year.

1400 is a Narishkeit festival year.

1200 is a leap year.

1080 is a leap year.
1080 is a Brimborion festival year.
```

6. Image Comparison

Program Name: Image.java Input File: image.dat

Bobby has made a new website. He originally had all of the images for his site on his computer. When he decided to host his site on the cloud, he had to transfer all his images to the cloud. Unfortunately, the cloud company changed the image format and names of the files to compress them, so now Bobby has to figure out which images are which. Unfortunately, with the compression algorithm, some of the pixel colors have changed slightly from the ones on his computer.

As a result, if Bobby wants to figure out which picture on the server corresponds to one on his computer, he needs to compute a difference, or "diff", between each image on the server, and the one on his computer. The correct picture on the server is the one with the smallest diff. The total diff score for the pair of images is the sum of the differences of each pixel. Each pixel is composed of a red, green, and blue channel, with a min value of 0 and a max value of 255 for each channel. Color values are traditionally stored in hexadecimal, or base 16, so they will range from 00 to FF. And as a result, one pixel consists of 6 hexadecimal digits, where the first two are the value of the red channel, the second two are the value of the green channel, and the third two are the value of the blue channel.

Input

The first line of input will be an integer T that is the number of cases to follow. The first line of input in each case contains three integers, N, W, and H, representing the total number of images on Bobby's site, and the width and the height of the images. All the images are of the same width and height.
2N images will follow. The first N images are the images from Bobby's computer, and the second N images are the images from the cloud server.
Each image consists of H + 1lines. The first line contains the filename of the image. The next H lines each contain W space-separated pixel values. (example: ffac1d dc23e6 b5ad4b)

Output

For each original image on Bobby's Computer, output the name of that image, followed by a colon and a space, followed by the name of the closest image on the cloud server. There will always be exactly one closest image. Place a blank line between each case.

Constraints

```
1 <= T <= 10
1 <= N <= 10
1 <= W, H <= 20
```

Example Input File

```
2
2 1 1
icon.png
ff89a3
background.png
666666
X3F.jpg
626163
AB.jpg
f190b2
3 2 2
circle.png
dd00ff c56a88
ee00ff 32684a
square.png
a3bcff ce3a82
21aafd 621aa1
triangle.png
3e00ff 4e29a8
3ea0ef 2684a3
5YaS.jpg
aabc3f c9318a
2aaf4 621aa5
H7rT.jpg
3a02f1 4e2aaa
39a1e1 2681a6
P77d.jpg
da00f9 c56b87
eb00fa 32604a
```

Example Output to Screen

```
icon.png: AB.jpg
background.png: X3F.jpg

circle.png: P77d.jpg
square.png: 5YaS.jpg
triangle.png: H7rT.jpg
```

Explanation of Output

To find which image is closest to icon.png, we do a diff between icon.png and X3F.jpg, and icon.png and AB.jpg, and see which image has the lower difference.

There is only 1 pixel, so the difference is just the difference of that pixel. For X3F.jpg, the difference is |0xFF-0x62| + |0x89 – 0x61| + |0xA3-0x63| = 157 + 40 + 64 = 261. For AB.jpg, the difference is |0xFF – 0xF1| + |0x89 – 0x90| + |0xA3 – 0xB2| = 14 + 7 + 15 = 36. 36 is less than 261, so AB.jpg is closer than X3F.jpg, and since there are only 2 images, AB.jpg is the closest. Therefore the closest image to icon.png is AB.jpg, and we can assume that AB.jpg is the corresponding image on the server.

7. Lorem Ipsum

Program Name: Lorem.java Input File: lorem.dat

In order to graduate, Joel has to pass the history class he is in. To pass his history class, Joel has to submit 9 different papers during the semester. Joel spent hours working on each of the papers, but realized that he always seemed to get the same grade no matter how much effort he put in. It dawned on him that none of the TA's actually read any of the student's papers and instead arbitrarily assigned grades. In order to test this hypothesis, Joel wants to submit a fake paper. He knows that just printing out a bunch of random filler text would stand out too much, so Joel wants to generate text that looks closer to English. He has thought about using a Markov chain text generator.

The basic premise is to first analyze a piece of known text and determine the probabilities of each word appearing after the previous *n* words and then generate the random text by choosing a random seed and selecting the words one at a time based on the previous *n* generated words.

For example, if the word *sunset* occurs thrice (three times) as often as the word *man* after the word *beautiful* (and no other words appear following *beautiful*), then if we use n=1, the probability of *beautiful sunset* is 0.75, while the probability of *beautiful man* is only 0.25.

Input

The first line of input contains an integer C that is the number of cases to follow. The first line of input in each case contains T, the number of sentences in the source text. The next T lines each represent a single sentence of the known text Joel wants to analyze and use for likeness. A sentence is a series of space delimited lowercase words. Assume the sentences are not connected, so the last word of a sentence does not lead to the first word of the next sentence.

Output

For each test case, print the probabilities of all possible word pairs. That is, for each unique word, print every word that can follow it and the respective probabilities, rounded to the nearest hundredth. If no words follow it, then omit the word. The lines should be sorted lexicographically by the unique word on that line. The listing for each word should be sorted by probability of occurrence and then also lexicographically if two words have equal probability.

Constraints

```
1 <= T <= 50
1 <= Number of unique words <= 100
```

Example Input File

```
3
3
a beautiful sunset that was mistaken for a dawn
that was a beautiful sunset
a beautiful man
4
if ever i would leave you
i would never leave you today
if i leave will you leave too
if i will leave you leave too
2
she loves me she loves me not she loves me she loves me not
she loves him more than she loves me or she loves you
```

Example Output to Screen

```
a: 0.75,beautiful 0.25,dawn
beautiful: 0.67,sunset 0.33,man
for: 1.00,a
mistaken: 1.00,for
sunset: 1.00,that
that: 1.00,was
was: 0.50,a 0.50,mistaken

ever: 1.00,i
i: 0.50,would 0.25,leave 0.25,will
if: 0.67,i 0.33,ever
leave: 0.50,you 0.33,too 0.17,will
never: 1.00,leave
will: 0.50,leave 0.50,you
would: 0.50,leave 0.50,never
you: 0.67,leave 0.33,today

him: 1.00,more
loves: 0.71,me 0.14,him 0.14,you
me: 0.40,not 0.40,she 0.20,or
more: 1.00,than
not: 1.00,she
or: 1.00,she
she: 1.00,loves
than: 1.00,she
```

Explanation of Output

In the first example the word *a* was thrice followed by *beautiful* and once followed by *dawn*, so the probability of *beautiful* following *a* is 0.75, while *dawn* following *a* is only 0.25. The words *dawn* and *man* are never followed by another word so they are omitted in the example output

In the second example on the second line *leave* and *will* both have a probability of .25 so they are printed lexicographically left to right.

8. Mining

Program Name: Mining.java Input File: mining.dat

Your spaceship has landed on an alien planet, and you have discovered a cavern with valuable crystals in it. Given that each of these crystals will more than pay for your trip if you were to sell them back home, you want to collect as many as possible.
Your radar has given you a full 2-D map of the cavern. There are some sections that are already empty, marked with a period '.'. Some sections are made of rock, but drillable using the drill that came with your ship, and are marked with 'X'. All other sections are made of rock that is too hard to drill through, and they are marked with 'O'. Given that you can drill as much of the drillable rock as you want, how many crystals can you collect in the scanned area? Your mining team will start from an area somewhere on the map marked 'S', which is empty. Each space with a crystal is marked 'C', and is also empty except for the crystal. You do not want to drill or move outside the radar area, because there could be unstable rock formations there, causing the death of you and your crew. You also cannot move diagonally within the map.

Input
The first line of input contains T, the number of test cases that follow.

The first line of each test case contains two integers N and M, the number of rows and columns of the radar scan.
The next N lines each contain M characters, designating that row of the radar scan.

Output
For each test case, print the maximum number of crystals you could collect by any amount of drilling.

Constraints
```
1 <= T <= 15
1 <= N, M <= 15
Number of 'S' = 1
Number of 'C' >= 0
```

Example Input File
```
4
1 2
SC
1 3
SXC
1 3
SOC
3 3
SOC
XOO
C.C
```

Example Output to Screen
```
1
1
0
2
```

Explanation of Output
In the first case, a crystal was right next to your starting area with nothing in the way.
In the second case, a crystal was right next to your starting area. There is rock between you and it, but it is drillable, so you can still reach the crystal.
In the third case, a crystal was right next to your starting area. There is rock between you and it, and it is not drillable, so you cannot reach the other crystal.
In the fourth case, you can drill through the rock below you to reach the bottom crystal and across the empty space to another, but the top crystal is entirely separated by a wall of rock that is not drillable, so there is no way to reach it.

9. Odd Numbers

Program Name: Odd.java Input File: odd.dat

The 3rd grade students in Mrs. Johnson's class are learning about odd and even numbers. To reinforce the concept, Mrs. Johnson had the students write the odd numbers an odd number of times per line like so:

```
 1
 3  5  7
 9 11 13 15 17
19 21 23 25 27 29 31
```

Notice the first line has 1 number, the next line 3, the next 5, and so forth containing all odd numbers in sequential order. Given the number of odd numbers in a line, you are expected to write a program that will give the sum of the last 3 numbers of that line.

Input
The input is a sequence of lines, each having a single odd number N, where ($1 < N < 100$). N denotes the number of odd numbers in that particular line.

Output
For each value of N, you will print the sum of the last 3 odd numbers in that line.

Example Input File
```
3
5
7
```

Example Output to Screen
```
15
45
87
```

10. Permutations

Program Name: Permutations.java Input File: permutations.dat

A permutation of a word is one in which the letters have been shuffled but the set of letters do not change. Given a word, find out how many distinct permutations of the word there are. If two permutations are lexicographically equivalent, they are not distinct.

Input
The first line of input contains T, the number of test cases.

The next T lines each represent a test case. Each test case consists of a single alphabetical lowercase word.

Output
For each test case, print the number of distinct permutations of the word there are.

Constraints
```
1 <= T <= 50
1 <= number of permutations <= 10^9
```

Example Input File
```
3
tall
splat
bug
```

Example Output to Screen
```
12
120
6
```

Explanation of Output
The word *tall* has the following 12 distinct permutations: *tall, tlal, atll, altl, ltal, latl, lalt, ltla, llat, llta, allt, tlla*.

11. Thanksgiving

Program Name: Thanks.java Input File: thanks.dat

Cooper and his colonists have landed on a new planet to settle, and Thanksgiving is just around the corner. Cooper wants to keep Thanksgiving as close to the old Earth as possible, so he tries to find alien creatures similar to turkeys. He ends up finding a couple that have arms like wings, legs like drumsticks, and meat like turkey breast meat.
To satisfy his colonists, he takes each of their orders for Thanksgiving, which consists of the number of wings they want, the number of drumsticks they want, and the number of pounds of breast meat they want. After doing this, Cooper finds several species of alien creatures with similar enough physical similarities. Each fully grown adult alien has a number L of legs that are similar to drumsticks, a number A of arms that are similar to wings, and B pounds of breast meat.
Since it is really hard to breed, raise, and feed an alien species, Cooper only wants to choose one to domesticate. He wants to choose the species that he will have to raise the least number of to fulfill the Thanksgiving order. Which species should he choose?

Input
The first line of input contains T, the number of test cases that follow.

The first line of each test case will be a single integer C, the number of colonists in Cooper's colony. The next C lines describe the Thanksgiving order of each colonist. Each line contain 3 space separated integers: cL, cA, and cB, the number of each type of meat the colonist wants for Thanksgiving dinner.
The next line of each test case contains a single integer N, the number of suitable species Cooper has found.
The next N lines describe the meat produced from each species. Each line contains a string, the name of the species, and then three space separated integers: L, A, and B, as described above.

Output
For each test case, print the name of the species that fulfills the Thanksgiving order with the least number of individuals of that species. It is guaranteed that at least one species will be able to fulfill the order, and that there will be a unique "best" species.

Constraints
```
1 <= T <= 10
1 <= C, N <= 10
1 <= cL, cA, cB, L, A, B <= 1000
```

Example Input File

```
3
2
0 0 1
0 0 2
2
A 0 0 2
B 0 0 1
3
2 4 3
1 3 7
0 2 3
4
A 1 2 1
B 0 2 1
C 1 4 3
D 0 1 1
2
5 5 5
6 3 8
3
A 1 2 1
B 1 3 3
C 1 4 2
```

Example Output to Screen

```
A
C
B
```

Explanation of Output

In the first case it seems the species on this planet appear to be snake-like, with no arms and legs. Thankfully, the total Thanksgiving order is 0 wings, 0 drumsticks, and 3 pounds of meat. If Cooper were to domesticate species A, it would take 2 A's to fulfill the Thanksgiving order. If Cooper were to domesticate species B, it would take 3 B's to fulfill the Thanksgiving order. Therefore, since it would take less A's than B's, the answer is A. In the second case, there is a wider variety of animal species available from which to choose. Again, a comparison of the settlers' orders to the animals available tells us that animal C is the best choice. The third case ends up with B as the favorite.

12. Tic Tac Toe

Program Name: Tic.java Input File: tic.dat

Tic Tac Toe is a two player game where the players alternate drawing X or O onto a 3x3 grid. The player who succeeds in placing three X's or three O's in a horizontal, vertical, or diagonal row wins the game. If no empty spots remain in the grid and no player has won, then the players draw.

Given the layout of a grid, determine if the game is ongoing, a win, or a draw.

Input
The first line of input contains T, the number of test cases.

Each test case consists of 3 lines representing a grid. Each of the 3 lines in the grid has 3 characters. `O` represents the circle player, `X` represents the cross player, and `.` represents an open space.

Output
For each test case, print whether the game is ongoing, a win, or a draw. If the game is a win, print the player that won. You have four possible outputs – X WINS, O WINS, DRAW, and ONGOING.

Constraints
```
1 <= T <= 10
```

Example Input File
```
3
OOO
X.X
XXO
...
.O.
...
OXO
OXX
XOO
```

Example Output to Screen
```
O WINS
ONGOING
DRAW
```