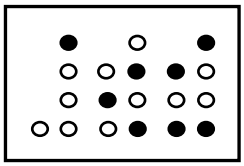
1. Binary Clock

Program Name: Binary.java Input File: binary.dat



Ms. Lane has a binary clock in her classroom. The clock has six columns of lights as shown to the right. Each light represents a power of two with 2^0 being the light on the bottom of a column and continuing upward to 2^3. The solid circles represent lights that are on and the empty circles indicate lights that are off.

The first column indicates the digit 0 or 1 and the second column indicates the digit 0 through 9 and the two columns together indicate the number of hours 00 though 12. The third column indicates the digits 0 through 5 and the fourth column indicates the digits 0 through 9 and the two columns together indicate the number of minutes 00 though 59. The fifth column indicates the digits 0 through 5 and the sixth column indicates the digit 0 through 9 and the two columns together indicate the number of seconds 00 though 59.

In the clock above, the time is `08:25:59` in hours, minutes, and seconds.

Since some of her students have trouble reading the clock, she wants you to write a program that will read the clock for the students.

Input

The first line of input will contain a single integer `n` that indicates the number of clock times to be read. For each clock time, there will be a matrix of 4 rows and 6 columns. The matrix will contain spaces that indicate that there is no light, `o`'s that indicate that the light is off, and asterisks (`*`) that indicates the light is on.

Output

For each clock time, you will print the time in the form `hh:mm:ss`, one clock time per line.

Example Input File

```
2
 * o *
 oo**o
 o*ooo
ooo***
 o o o
 o*o**
 *o*o*
*o**o*
```

Example Output to Screen

```
08:25:59
12:53:47
```

2. Cell Phone

Program Name: Cell.java Input File: cell.dat

Andrea is planning to buy a new cell phone and is looking at several cell phone plans. One of the plans offers two different "per minute" plans. Plan A costs more per month than Plan B but less per minute used. The major difference in the two plans is the way their "per minute" time is calculated. For any call over an integral number of minutes, Plan A truncates the seconds while Plan B rounds the length of the call up to the next minute. For example, a phone call that lasts 3 minutes and 12 seconds would be charged for 3 minutes with Plan A and 4 minutes with Plan B.

Andrea has the lengths of all her phone calls for the last month. You are to write a program for her that will calculate the number of minutes for which she would be charged under each plan.

Input

The input file contains an unknown number of lines. Each line of input will be in the form `m.s` where `m` represents the number of minutes and `s` represents the number of seconds of a phone call.

Output

You will print the number of minutes for which she would be charged under Plan A and under Plan B in the format shown below.

Example Input File

```
14.5
12.0
13.2
3.56
4.12
0.8
1.1
2.0
9.29
2.0
1.1
```

Example Output to Screen

```
PLAN A: 61
PLAN B: 69
```

3. Change You Can Believe In

Program Name: Change.java Input File: change.dat

Libby's favorite hobby is shopping. One day, when she was shopping, Libby bought some shoes that cost $53.38. Libby handed the clerk $54.00 and received 2 quarters, 2 nickels, and 2 pennies in change. Libby wondered why she was not paid with a half-dollar, a dime, and two pennies. Then, she began to wonder how many other ways the 62 cents could have been paid. You are to write a program that will determine how many different ways a given amount of money could be paid using only pennies, nickels, dimes, quarters and half-dollars.

Input

The first line of input will contain a single integer `n` that indicates the number of test cases to follow. Each of the following `n` lines will contain a single integer `m` ($0 < m < 100$), that indicates an amount of change to be given.

Output

For each test case, you will print the number of different ways the change could be paid.

Example Input File

```
4
62
48
17
87
```

Example Output to Screen

```
77
39
6
187
```

4. Elite Numbers

Program Name: Elite.java Input File: elite.dat

The sieve of Eratosthenes is a well known process for finding the prime numbers by the following process:

- List all of the natural numbers from 1 to infinity.
- Starting with the number 2, do not mark out the number 2 but do mark out every second number thereafter.
- Then, starting with the number 3, do not mark out the number 3 but mark out every third number thereafter
- Continue this process forever ...

This process will leave only the number 1 and all of the prime numbers unmarked.

The program you will write will be similar to the sieve of Eratosthenes but will leave only what we will refer to as Elite numbers. The process is as follows:

- List all of the natural numbers from 1 to infinity.
- Remove all even numbers leaving the odd numbers:
  1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, ...
- The 2nd number is 3. Start counting with the number 1 and remove every 3rd number leaving:
  1, 3, 7, 9, 13, 15, 19, 21, 25, 27, 31, 33, ...
- The 3rd number is 7. Start counting with the number 1 and remove every 7th number leaving:
  1, 3, 7, 9, 13, 15, 21, 25, 27, 31, 33, ...
- Continue this sieve with the 4th number which is 9.
- Then continue this process forever …

Input

The first line of input will contain a single integer `n` that indicates the number of test cases to follow. Each of the next `n` lines will contain a single natural number smaller than 1500.

Output

For each natural number input, you will print `ELITE` if it is an Elite number or `NOT ELITE` if it is not an Elite number.

Example Input File

```
4
33
85
1053
500
```

Example Output to Screen

```
ELITE
NOT ELITE
ELITE
NOT ELITE
```

5. Family Portrait

Program Name: Family.java Input File: family.dat

Roger is a software developer. Sometimes, he has to rotate pictures for his applications manually from portrait layout to landscape, one pixel at a time, so he can print the picture on a given printer. You are to write a program that will do this rotation for him.

Input
The first line of input will contain a single integer `n` that indicates the number of pictures he needs to rotate. For each picture, there will be 18 rows of pixels with 12 columns of pixels in each row representing a picture that is 2 inches wide and 3 inches long. There will be a blank line after each picture. Each pixel will be represented by a keyboard character.

Output
For each picture input, you will print the picture after it has been rotated clockwise 90 degrees. Print a blank line after each picture printed.

Example Input File
```
1
1BBBBBBBBBB2
1AAA5678GGG2
123488889012
18888**88882
12888**88882
12777**77712
1234****9012
1234****9012
12888..88812
18888..88882
1((((__))))2
1(((__.__))2
1---..-..--2
1$$$$$$$$$$2
133556677882
1444aaa55aa2
1eerrttyyuu2
1BBBBBBBBBB2
```

Example Output to Screen
```
111111111111111111
Be43$-((82222282AB
Be43$-((88337883AB
Br45$-((88447884AB
Bra5$._(88**78885B
Bta6$.__..*****86B
Bta6$-._..*****87B
By57$._)88**78888B
By57$._)88997889GB
Bua8$-))88007880GB
Bua8$-))81111881GB
222222222222222222
```

6. Hash Codes

Program Name: HashCodes.java Input File: hashcodes.dat

Hash codes are used to map items to a numerical value by using some mathematical formula. Once mapped, there are quick retrieval methods to locate the item for use. The bad thing about using hash codes is that sometimes, more than one item will map to the same hash code thus causing a collision.

You are to write a program that will determine the number of collisions that could happen given the following parameters:

- The items to be mapped will be composed of a given number of upper case letters of the alphabet.
- The letters used in any word will be in increasing order of the alphabet.
- The items will be mapped by finding the sum of the locations of the letters in the alphabet. A=1, B=2, C=3, etc.)

Example: There are 7 three letter words with letters in ascending order that have a hash code of 12. They are:

```
ABI
ACH
ADG
AEF
BCG
BDF
CDE
```

Note: "Words" are combinations of letters that meet the criteria listed above.

Input

The first line of input will contain a single integer `n` that indicates the number of cases to consider. Each of the following `n` lines will contain two integers in the form `w h` where $2<w\leq5$ is the numbers of letters in the word and `h` is the value of the hash code.

Output

For each case that is input, print the number of collisions that could happen.

Example Input File

```
3
3 12
4 22
3 65
```

Example Output to Screen

```
7
34
14
```

7. Interlocked

Program Name: Interlocked.java Input File: interlocked.dat

Two strings are said to be "interlocked" to a third string if:

- All of the characters in the first string are contained in order, but not necessarily consecutively, in the third string.
- All of the characters in the second string are contained in order, but not necessarily consecutively, in the third string.
- Letters that appear in both words do not have to appear twice in the interlocked word.
- Letters that appear more than once in a single word must appear more than once in the interlocked word.

You are to write a program to determine if two strings are "interlocked" in a third string.

Input

The first line of input will contain a single integer `n` that indicates the number of data sets to follow. Each of the following `n` lines will contain a list of three words composed of lower case letters of the alphabet and each separated by a single space. The first two words are the words for you to determine if they are interlocked in the third string.

Output

For each set of three words you will print `YES` on a single line if the first two strings are interlocked in the third string and `NO` if they are not.

Example Input File

```
4
bit one bointe
bit one bioonaet
bit one bierontt
bit byte bieyonte
```

Example Output to Screen

```
YES
YES
NO
YES
```

8. Median Mile

Program Name: MedMile.java Input File: medmile.dat

James is keeping the times in the mile race for the swim team at his school. Before selecting the swimmers for the district swim meet, the coach wants to know each swimmer's median time from the mile swims that she has swum this year.

The median time is:

- the middle time of an odd number of times sorted from least to greatest.
- the average of the two middle times of an even number of times sorted from least to greatest.

Input

The first line of input will contain a single integer `n` that indicates the number of swimmers to be considered. Each of the following `n` lines will contain the swimmer's first name with no spaces followed by a list of times in the form `mm:ss.hh` where $15 \leq$ `mm` ≤ 40 is the number of minutes, $00 \leq$ `ss` ≤ 59 is the number of seconds, and $00 \leq$ `hh` ≤ 99 is the number of hundredths of seconds. Each of the times will be separated by a single space.

Output

For each swimmer, you will print the student's name and a space followed by her median time in the format `mm:ss.hh`. The hundredths should be truncated, not rounded.

Example Input File

```
3
MARY 21:12.32 22:14.45 23:55.84 24:48.99
ANN 28:16.39 28:10.87 27:04.00 27:05.01 29:29.71
LUCY 33:43.82 28:24.95 30:52.94 29:55.26
```

Example Output to Screen

```
MARY 23:05.14
ANN 28:10.87
LUCY 30:24.10
```

9. Nautical Miles

Program Name: Nautical.java Input File: nautical.dat

Philip goes on a lot of cruises. At the noon update, the ship's captain reports the distance that the ship has traveled since the last port and the distance left to travel to the next port in terms of nautical miles. You know that one nautical mile is approximately 1.15 miles in our English measurement system.

You are to write a program that will convert nautical miles to English miles.

Input
The first line of input will contain a single integer `n` that indicates the number of nautical mile distances to be converted. Each of the following `n` lines will contain a single positive integer less than 50,000.

Output
For each integer input, you will print on a single line the number of English miles, rounded to the nearest tenth, represented by that integer.

Example Input File
```
3
1024
926
877
```

Example Output to Screen
```
1177.6
1064.9
1008.6
```

10. Runners

Program Name: Runners.java Input File: runners.dat

Russell, Seth, and Thomas are runners for their school's programming contest. Their job is to collect the disk containing a program from a programming team and deliver it to the judge's room for judging. They noticed that frequently the person with the shortest path to the programming team was not always the person that picked up the program. They want a program that will find which runner would have the shortest path to a programming team given the location of each runner, the team that has a program complete, and the setup of the contest room.

You are to write a program that will find the runner with the shortest path from his original location to the team with the finished program to the judge's room.

The rectangular contest room will contain the following symbols:

`.`	Indicates an empty floor space.
`*`	Indicates a wall or desk that the runner must go around.
`P`	Indicates the location of the programming problem. A runner must move either horizontally or vertically adjacent to the cell containing the `P` to collect the problem.
`R`, `S`, and `T`	Indicates the locations of `RUSSELL`, `SETH`, and `THOMAS` respectively.
`J`	Indicates the location of the entrance to the judge's room.

The rules for movements are:

- All movement must be vertical or horizontal.
- A runner cannot go through a wall, another runner, or the location of the programming problem.
- The length of the runner's path does not include the original location of the runner or the location of the programming problem but does include the location of the entrance to the judge's room.

Input

The first line of input will contain a single integer `n` that indicates the number of contest rooms to follow. For each contest room, the first line will contain two integers in the form `r c`, indicating the number of rows and columns in the contest room, respectively. The next `r` lines will contain the maze with symbols listed above with no spaces.

Output

For each contest room, you will print on a single line the length of the shortest path from the runner's location to the programming problem to the judge's room followed by a space and the name of the runner with the shortest path. If two or more runners have a path that is a shortest path, list those runners in alphabetical order separated by a space. Names should be all uppercase.

Example Input File

```
2
5 12
************
*.....R*T.**
**P**....S**
*.........**
*****J******
5 10
**********
**S*T...**
*........J
*..P*R..**
**********
```

Example Output to Screen

```
10 RUSSELL
8 SETH THOMAS
```

11. Threes and Zeroes

Program Name: Threes.java Input File: threes.dat

Given a positive integer less than 50, there is a multiple of that integer that can be written using only the digits 3 and 0. For example, 300 is the smallest multiple of 4 that contains only the digits 3 and 0.

You are to write a program that will find the smallest multiple of a given integer `x` that contains only the digits 3 and 0.

Input
The first line of input will contain a single integer `n` that indicates the number of test cases to follow. Each of the following `n` lines will contain a single integer `x` ($3 < x < 50$).

Output
For each number `x` input, you will print on a single line the smallest multiple of `x` that contains only the digits 3 and 0.

Example Input File
```
4
4
5
12
38
```

Example Output to Screen
```
300
30
300
330030
```

12. Tree Decompression

Program Name: TreeDecompression.java **Input File: treedecompression.dat**

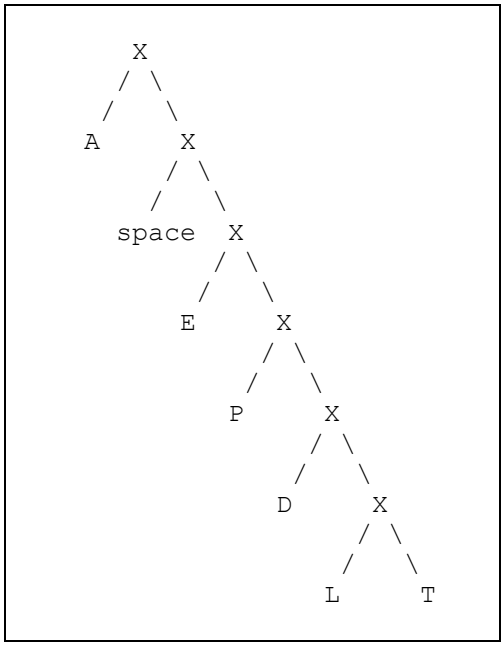
There are various methods for data compression that trade-off speed of compression and/or decompression with compression size. A relatively simple form of text compression, which we will call Tree Compression, involves using small bit sequences to represent the unique characters in a string and then creating a unique sequence representing each string.

The steps for this tree compression are:

- Count all of the unique characters in the string.
- Sort the characters by the number of times they appear in the sequence from most occurring to least occurring. Characters that occur the same number of times should be sorted by their ASCII numeric value.
- To insure that no two characters begin with the same bit sequence and to use as few bits as possible, assign a binary sequence to each character in your sorted list using the following method:
  - The last character in your list will always be all 1's.
  - If there is only a single character is your list, that character is assigned a 1.
  - Otherwise, if there is more than one character in the list, the first character in the list is assigned the binary value 0.
  - The second character, unless it is the last one, would be assigned 10.
  - The third character, unless it is the last one, would be assigned 110.
  - Continue this process until finally, as stated above, the last character has a 1 at the end instead of 0, i.e., it would be all 1's.

For example, for the string `ADA ATE APPLE` containing 4 `A`'s, 1 `D`, 1 `T`, 2 `E`'s, 2 `P`'s, 1 `L`, and 2 spaces, the encoding for each character is shown in the chart to the right.

Char	encoding
A	0
Space	10
E	110
P	1110
D	11110
L	111110
T	111111



By reading bits in, one by one, we can tell what character it is since all strings are unique and all sequences are unique from left to right.

One way to figure out what character you are reading is to put the characters into a tree. You start at the top of the tree, and for every bit you move either left for a 0, or right for a 1. When you complete a character, you start again at the top with the next bit. A tree for this example would be as shown in the chart to the left (the X's are just nodes).

Note: some strings require more bits with this compression scheme than if they were in their standard 8-bit ASCII form.

12. Tree Decompression (cont.)

Input

The input file will contain an unknown number of lines, each one containing a string of 1's and 0's which represent the bits of a compressed string. The string will either be in Tree Compression format, or in standard ASCII format, depending on the value of the first byte.

If the first byte is not 0, then the value of that byte is the number of unique characters in the string. Then, there will be a byte for each unique character, in the order the characters appear in the decompression tree. The rest of the string will then be the compressed string representation.

In example 1 below, the first byte is 3 indicating that there are three bytes to follow as they would appear in the decompression tree. Following those three bytes is the compressed string.

```
00000011 01000001 01000110 01001100 01110011100
   3        A        F        L     compressed string
```

If the first byte is 0, that means the rest of the string is just a series of bytes that represent ASCII characters in the order they appear in the string. In other words, the string isn't compressed.

In example 2 below, the first byte is zero followed by the corresponding ASCII equivalents.

```
00000000 01000001 00100000 01000100 01001111 01000111
   0        A      space      D        O        G
```

Output

For each input line you will print the ASCII version of the string that is represented, one line for each line of input.

Example Input File

```
00000011010000010100011001001100011100111100
00000000010000010010000001000100100111101000111
```

Example Output to Screen

```
ALFALFA
A DOG
```