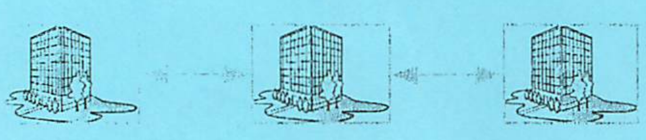
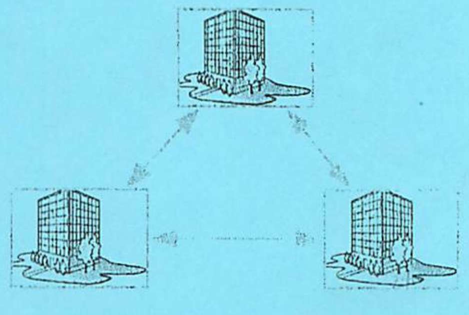
Let's Get Critical

You manage the wide-area-network that allows field offices in your company to communicate with one another. Each office has some number of direct connections to other offices, and two offices can communicate if there is a sequence of direct connections that lead from one office to the other. You've been asked to verify that all offices have redundant communication paths with every other office. The way you decide to do this is to detect any *critical* offices in the network. An office is critical if removing it, and all its direct network connections, would leave any remaining pair of offices unable to communicate. For example, the network on the left has a critical office because removing the middle office (and its direct connections with other offices) would leave the two end offices with no communication path between them. The network on the right has no critical office, indicating that every pair has redundant communication paths.





Your program should read from the file "critical.in". The file describes multiple networks as a sequence of whitespace-separated integers. The first integer tells how many networks are described in the file. Each network is then described as a positive integer *n* telling how many offices are in the network, an integer *m* telling how many direct network connections are in the network, and then *m* pairs of integers, each indicating a pair of offices with a direct network connection. Offices will be designated as integers in the range 1 . . . *n*. No pair of offices will be listed more than once per network. No office will be listed as having a direct connection with itself. Connections are symmetric, meaning traffic can flow in both directions.

For each network, your program should output a single line which lists all critical offices (as integers in increasing order with a space between each), outputs 0 to indicate no critical offices exist, or outputs -1 to indicate that the given network has a pair of offices that cannot communicate (ie, there's not even one communication path between some pair of offices).

Example

On input:

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

The following output would be accepted:

```
2 3
0
-1
```

Parallel Plotting

In this problem you are to print several rows of side-by-side figures using character '*'. In the file "pplot.in", the first line contains a positive integer indicating the number of rows of figures to be produced. Each subsequent line contains a sequence of positive integers indicating the sizes of side-by-side figures you are going to print in a row. For an even number, you should print a square and for an odd number, you should print a right triangle facing to the right. The number indicates the height and width of the figure, measured in '*' characters. Figures should be separated by exactly one column and rows of figures should be separated by exactly one blank line (as shown below).

Example

On input:

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

The following output would be accepted:

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

                 **********
                 **********
                 **********
      *          ********** *
      **         ********** **
*     ***        ********** ***
**    ****       ********** ****
***   *****      ********** *****
****  ******  ** ********** ******
***** ******* ** ********** *******

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

Change for a Dollar?

On a slow day at your frozen banana stand on Balboa Island you begin to wonder: Just how many ways are there to give change for a dollar? Obviously you could give four quarters, but you could also give three quarters, two dimes and a nickel, or even one hundred pennies. As a customer bursts your daydream with an order of "medium with almonds please", you promise yourself that as soon as you get home, you're going to write a computer program to answer this fascinating question.

Write a program that reads input from "change.in". The first line is a positive integer indicating how many problem instances are described in the file. Each subsequent line describes one instance of the problem and begins with a positive integer, which is the total value of change that needs to be given. Next on each line is a positive integer *n* indicating how many different coin values exist in your currency. Finally, on each line are *n* distinct positive integers, which are the coin values available for making change. All integers are separated by one space. For example, the problem described in the first paragraph (assuming you have pennies through half-dollars available) could be described as: `100 5 1 5 10 25 50`. Don't consider paper money, this problem is only about using the given coin values to make change.

For each problem, your program should output a single line with the number of different sets of coins that sum to the target amount. Note that the order in which the coins are given is irrelevant: $\{1, 1, 5\} = \{1, 5, 1\} = \{5, 1, 1\}$ are all equivalent sets.

Example

On input:

```
3
1 5 1 5 10 50 25
10 5 1 5 10 25 50
5 2 1 2
```

The following output would be accepted:

```
1
4
3
```

In It?

In the game of Scrabble, you are given seven tiles, each inscribed with a letter and a point value, and you try to find the highest-scoring word you can using any subset of your letters. To help your Scrabble game you write a program that takes seven letters and a word and tells you whether the word can be formed using some subset of the letters and the score the word is worth.

Your program should read from the file "init.in". The first line of the file has a positive integer indicating how many letters-word pairs the program should process. Each subsequent line of the file contains two strings. The first is seven characters indicating the letters you can use to form the word, and the second is the candidate word. As in Scrabble, letters may repeat in your list of letters and there may be one or more blank letter which can be used as a substitute for any letter and is indicated as an underscore. The word can be formed from the list of letters if a subset of the list of letters can be arranged to form the word. A blank letter can be used as a substitute for any letter in the forming of a word.

For each letters-word pair, your program should output a single line listing the points the word is worth or zero if the word can't be formed by the given letters. The score a word produces is the sum of the points of each letter in the word. Each of a, e, i, o, u, l, n, r, s, t is worth one point; each of d, g is worth two; each of b, c, m, p is worth three; each of f, h, v, w, y is worth four; each k is worth five; each of j, x is worth eight; and each of q, z is worth ten. Blanks are worth nothing. If a word can be made in more than one way (ie, with or without blanks) list the maximum score possible. All letters in the file will be upper-case.

Example

On input:

```
3
ABCDEFG BAD
ABCDEFG BEEF
ABC_EFG BEEF
```

The following output would be accepted:

```
6
0
8
```

Stop the Aliens!

Since the aliens will be landing soon and taking over the world, you decide to try to slow them down by changing the way math is expressed. That way when they find your treasure trove of mathematical expressions they won't be able to make heads nor tails of it.

Your new method has a few changes from what we normally do. Instead of writing expressions the normal way, we are going to first write a long string of digits; second, express arithmetic using indexes into the long string of digits instead of actual numbers, and third throw out precedence rules. For example "`785423 3 + 1 * 5`" indicates that we should add the third and first digits in 785423 (indices start with 1 and are right-to-left), and multiply the result by the fifth digit of 785423. So the result is $(4 + 3) \times 8$, or 56.

Your program should read from the file "stopthealiens.in". The first line has an integer indicating how many expressions are in the file. Each subsequent line contains one expression. Each expression begins with a long string of digits and is followed by an alternating sequence of integers and operations (beginning and ending with integers). The integers are to be interpreted as indexes into the long string of digits, with the rightmost digit having index 1. For each expression, your program should output a single line with the resulting number you get when you replace all the indexes with the digit they represent and then evaluate the resulting mathematical expression from left to right, ignoring all operator precedence rules. Every token on every line will be separated by exactly one space. Each operation will be one of the following: `+ - * / ^`. Division is integer division, and `^` is exponentiation.

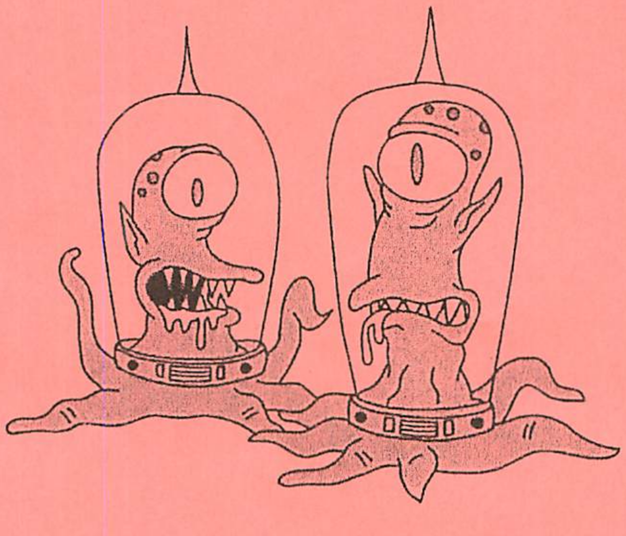
Example

On input:

```
2
785423 3 + 1 * 5
0987654321 4 + 8 / 5 ^ 3
```

The following output would be accepted:

```
56
8
```



THE TOMATO WORM OLYMPICS

If you have ever grown tomatoes, you have seen those big green worms that tomato plants attract. They are called "hookworms", or more commonly "tomato worms". People just think of them as an ugly nuisance, but actually they are very smart and have a complex social structure. They even have their own countries, with names like "Leafland", "Branchia", "Pulpica", etc. Each year those countries compete against each other in the "Tomato Worm Olympics" - abbreviated "TW Olympics".

By far the most popular event in the TW Olympics is *EATING*. The event has gotten so big that they need some help keeping records during the competition. The TW Olympic committee has asked you to write a program that will help them manage this years' competition. (Tomato worms still haven't invented computers, and they are painfully slow at using ours)

Competition takes place over a 3-day period in August, and competitors from around the world submit their eating totals for each day (telepathically) to the event judge. This year, FIVE countries are competing, and are numbered: (1) Leafland, (2) Manchewia, (3) Branchia, (4) Greenland, and (5) Pulpica. Each country has its own individual competitors (at most 50), each with its own integer assigned number (0-9999). An example file collected over the 3-day period would have this format:

```
-1
3 535 20
5 22 6
-2
3 535 8
1 400 16
3 221 7
1 400 6
-3
4 18 12
5 22 8
-999
```

Days 1, 2, and 3 are indicated with negative numbers (-1, -2, -3), once each and in that order, with -999 indicating the end of the competition and the end of the data. Individual results are listed one per line. In each individual result, the first number is the number of the country (1-5, as described above), followed by the worm's competitor ID number, and then the number of ounces that the worm ate. Worms can eat more than once – even multiple times per day. You can assume that there are only the 5 countries specified above, but each country can have up to 50 competitors. You can also assume that a particular worm's competitor ID number is always listed with the same country.

Your job is to print out the individual and team medal winners, based on total ounces eaten. The TW Olympics gives out Green, Red, and Brown medals for 1st, 2nd, and 3rd place, respectively. First print the team medals, with the medal, team number and name, followed by total ounces eaten, one per line. Than print the individual medals, with the individual worm ID, followed by that worm's total amount eaten, also one per line. For the above input example, your program should print out the medal results as follows (note the alignment after the colons[:], and the team numbers and names):

```
Team Green medal: 3 Branchia 35 oz
Team Red medal:   1 Leafland 22 oz
Team Brown medal: 5 Pulpica 14 oz
Worm Green medal: 535 28 oz
Worm Red medal:   400 22 oz
Worm Brown medal: 22 14 oz
```

Ties can be broken in any order. For example, if two teams tied for 2nd place, then either one can be given the Red medal, and the other would get the Brown medal. The input file is named "worms.in".

Card Counter

You have decided to become a notorious Las Vegas bigshot. With this goal in mind, you have done some research, and discovered that there is a lot of card-playing and gambling that goes on in Las Vegas (card games like Blackjack, Poker, etc.). You have also heard that in order to be effective as a card-player, it helps if you know how to *count cards*.

Card counting means keeping track of how many of each kind of card has already been dealt by the dealer. It is difficult and takes a good memory. You decide that is a lot of work, and so your plan is to write a computer program to count the cards for you. (You are also hoping that they will let you set up your laptop on the card table).

Your program reads characters from an input file "cards.in", one per line, each representing a card. The character "A" represents the "Ace" card; the characters "2" through "9" represent the number cards respectively; the character "0" represents the "10" card, and the characters "J", "Q", and "K" represents the "Jack", "Queen", and "King" cards. The character "E" means there are no more cards. The output of your program is a count of how many of each type of card has been read in. Output the counts in the order listed above. For example, if "cards.in" contained:

```
3
8
A
2
Q
3
A
K
Q
Q
E
```

Your program would output:

```
2
1
2
0
0
0
0
1
0
0
0
3
1
```

There were 2 "ace" cards

There was 1 "two"

There were 2 "threes"

There weren't any "fours"

etc....

There were 3 "queens", and 1 "king"

Note that regardless of how many lines are in "cards.in", there are always exactly 13 lines of output – one for each type of card.