X PROGRAMMING OLYMPIADS IN MURCIA 2012

Problem Set Index

Plan B for next SWERC

Background

As we all know, teams participating in ICPC or in any of the previous regionals receive one balloon for each problem solved. At the end of the contest, the team with more balloons wins.

As part of the selection process, the candidate members of our team for the next Southwestern Europe Regional Contest need to prove that they know how to handle our new secret weapon: an USB controlled laser cannon. In the highly unlikely case that our team got less balloons than some other team, they would use the laser cannon to burst other teams' balloons.

The Problem

We will assume that the contest takes place in a two dimensional room and that the only objects in the room are the target balloons and the laser cannon. Further, our laser has infinite range and the balloons are perfect circles. The cannon is in the lower left corner of the room (which has coordinates (0, 0)). Hence, to aim the cannon, the user only has to select the proper elevation angle.

Unfortunately, the laser cannon is a cheap model from ACME that can only fire at elevations from 0 sexagesimal degrees up to 90, in increments of 10 degrees. Also, it sometimes gets stuck when lowering the muzzle, hence if several shots are to be fired, it is important to fire them in order from lowest to higher elevation angle.

You have to write a program to calculate at which elevations the cannon has to be fired and how many balloons will be burst, given the size and coordinates of all the target balloons.

The Input

The input format is as follows:

An integer in a single line which says the number of problems to solve. Then, for each problem:

- An integer in a line of itself denoting the number of target balloons.
- One line for each balloon, with three integers giving the coordinates *(x, y)* of each balloon and its diameter (all distances are given in centimeters). No balloon is further than 15 meters from the cannon and all balloons are smaller than 1 meter in diameter.

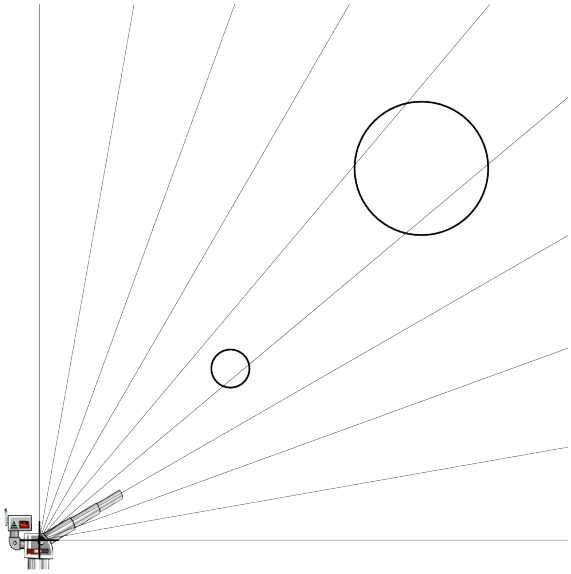
The Output

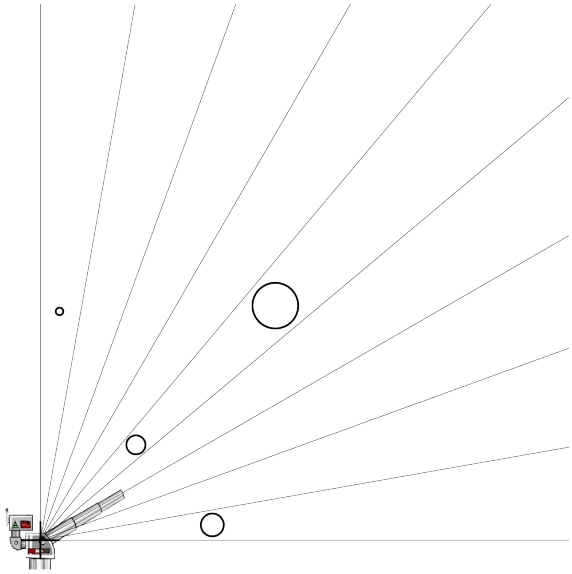
The output for each problem consists of one line for each shoot of the form “Fire laser at *d* degrees.”, where *d* is the elevation for the shot, followed by a line indicating how many balloons have actually been burst in the form “1 burst balloon.” when only one balloon has been burst, “No balloon burst.” if no balloon has been harmed and “*b* burst balloons.” if *b* balloons have been hit and $b > 1$.

Sample Input

```
2
2
100 90 20
200 195 70
4
123 123 25
90 8 12
50 50 10
10 120 4
```

The following figures are diagrams showing the positions of the balloons in the two sample problems and all the possible firing elevations:





Sample Output

```
Fire laser at 40 degrees.
2 burst balloons.
No balloon burst.
```

Bars

Background

Some things grow if you put them together.

The Problem

We have some metallic bars, theirs length known, and, if necessary, we want to solder some of them in order to obtain another one being exactly a given length long. No bar can be cut up. Is it possible?

The Input

The first line of the input contains an integer, t, $0 \leq n \leq 50$, indicating the number of test cases. For each test case, three lines appear, the first one contains a number n, $0 \leq n \leq 1000$, representing the length of the bar we want to obtain. The second line contains a number p, $1 \leq p \leq 20$, representing the number of bars we have. The third line of each test case contains p numbers, representing the length of the p bars.

The Output

For each test case the output should contain a single line, consists of the string YES or the string NO, depending on whether solution is possible or not.

Sample Input

```
4
25
4
10 12 5 7
925
10
45 15 120 500 235 58 6 12 175 70
120
5
25 25 25 25 25
0
2
13 567
```

Sample Output

```
NO
YES
NO
YES
```

Mirror codes

Background

Many kids “computers” use some kind of “punched cards” to select the game to play. The card reader has a variable number of switches whose state (on/off) is determined by the corresponding card state (hole/solid) at the switch position.

Thus, if the card reader has *n* switches, one could believe that the computer could have as much as 2^n games to play, but the actual number is a bit smaller. The reason for this is that all cards have two sides with one different game each.

6-bit card example

Card side A (code: 000111)

Card side B (code: 111000)

Binary codes that result in the same code when mirrored cannot be used, as there would be no way to distinguish what card side is being shown to the player, i. e., what game the player wants to play.

The Problem

We will generalize the above problem to find the number of different games that can be coded in a more modern computer whose reader consists of d digits of a different numeric base each. However, these base sequence shall always be mirror-invariable, so that any game code that gets mirrored should make sense for the card reader. In other words, if the reader has d digits whose bases are:

$b_1\ b_2\ b_3\ \cdots\cdots\ b_{d-2}\ b_{d-1}\ b_d$

It will always hold that:

$b_1 = b_d$
$b_2 = b_{d-1}$
$b_3 = b_{d-2}$

, and so forth.

Given a sequence of bases as described above, your program has to determine how many games can be coded so that they can be properly (and uniquely) determined by the number coded in a card that can be reversed to select a different game.

The Input

Each input case is represented by a single line composed by one first positive integer number that represents the number of digits (`d`), immediately followed by `d` integer numbers that represent the numeric bases (all of them > 1), as described above.

An input case with `d = 0` marks the end of the input. Obviously, this last case should not be processed (no output should be generated for it).

The Output

For each input case (but the last one that marks the end of input) your program should write a line with the number of games that can be coded in the computer, assuming that all games are to be played on a two-sided coded card that should represent a different game for each side.

No leading/trailing blank spaces should appear in these lines, and no blank lines should be written either.

You may assume that the solution for any input case will always fit in a 32-bit integer.

Sample Input

```
2 2 2
4 2 2 2 2
6 2 2 2 2 2 2
0
```

Sample Output

```
2
12
56
```

Tennis contest

Background

Nadal or Djokovic? Who is the best one?

The Problem

The two most famous tennis players, *A* and *B*, are facing each other in up to *2n-1* matches. The one who wins *n* matches will be the best player in the world. We supose the result of each game doesn't depend on the rest, and there is a constant likelihood, *p*, of *A* to win a match. Draw is an invalid result. Which is in advance the probability of *A* to win the title?

The Input

The first line of the input contains an integer, *t*, indicating the number of test cases. For each test case, two lines appear, the first one contains a number *n*, 1<=*n*<=25, representing the number of wins A has to reach. The second line contains a number *p*, 0<=*p*<=1, representing the probability of *A* to win a match.

The Output

For each test case the output should contain a single line with the number representing the probability in advance of *A* to win the title of best player in the world.

Sample Input

```
5
25
0.5
25
0.4
25
0.6
15
0.8
10
0.95
```

Sample Output

```
0.50
0.08
0.92
1.00
1.00
```

Oh, My Trees!

Background

Pablito likes trees of all kinds: palm trees, pine trees, orange trees, eggplants, etc. But his favorites are binary search trees.

As you should know, **binary search trees** (BST) are binary trees (i.e., each node of the tree can have zero, one or two subtrees, which are called *left* and *right* subtrees) with the following property:

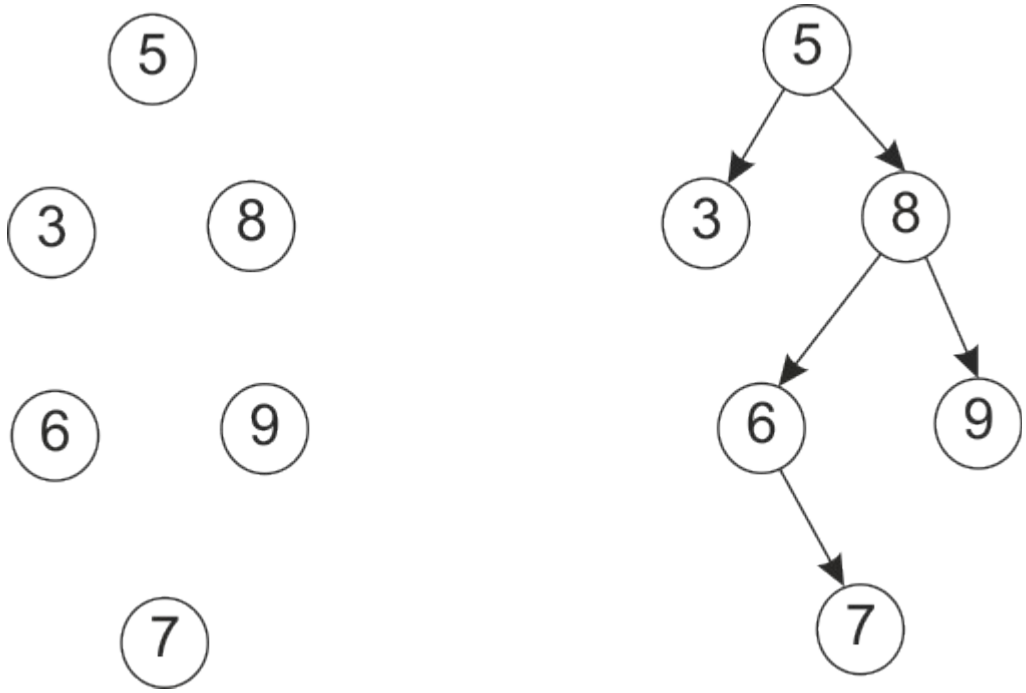
- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.

Pablito is very proud of his huge collection of BS trees. However, he has a younger brother who loves deleting trees. Poor thing! He is only 2 years old!

Pablito's brother has the ability of deleting only the edges of the trees, but not the nodes. Can you help Pablito to fix up the disaster made by his brother?

The Problem

We have a set of different integer values, which are the keys of a BST. But the edges of the BST have been deleted; we know the level (or *depth*) of each key in the tree, but we do not know the father-child relationships.



A sample case. Left: a BST with the edges deleted. Right: the same tree with the left-child / right-child relationships undeleted.

Your task is to recover the right-child / left-child relationships for all nodes.

Input

The input can contain different test cases. The first line of the input indicates the number of test cases.

For each test case, the first line contains an integer, *H*, which is the total hight of the tree (i.e., the number of levels). Then, there are *H* lines, each of them with a set of integers separated by spaces. Each line contains the keys of each level of the BST. You can assume that the description of the tree is always valid; keys are not repeated.

For example, the previous tree would be represented as:

```
4
5
3 8
6 9
7
```

Output

For each test case, you have to output each key in a line, in the same order as they appear in the input. For eack key, you have to indicate his children in the following format:

```
node:leftChild-rightChild
```

If the node does not have left or right child, no number is printed. For example, in the previous three the result should be:

```
5:3-8
3:-
8:6-9
6:-7
9:-
7:-
```

Sample Input

```
3
4
5
3 8
6 9
7
5
23
59
35
39
36
8
10
4 12
1 6 11 31
0 3 5 8 51
34
49
40
38 45
```

Sample Output

```
5:3-8
3:-
8:6-9
6:-7
9:-
7:-
23:-59
59:35-
35:-39
39:36-
36:-
10:4-12
4:1-6
12:11-31
1:0-3
6:5-8
11:-
31:-51
0:-
3:-
5:-
8:-
51:34-
34:-49
49:40-
40:38-45
38:-
45:-
```

Bees' ancestors

Background

Maya is a bee that likes to help her friends. Willy is a drone bee and Maya's best friend. He has just discovered that he has no father, and he is worried about that.

The drone bee Willy.

The Problem

Maya knows that female bees have two parents (one father and one mother), but male or drone bees have a mother but no father. This is because if an egg is laid by an unmated female, it hatches a drone bee, but if, instead, an egg was fertilized by a male, it hatches a female.

After Maya talks with Willy about this, he begins to think about the number of ancestors that he has. He has one mother. He also has two grandparents (one grandmother and one grandfather). And he has three great-grandparents. Since Willy is very lazy, he doesn't want to perform more calculations and he is asking you to write a program that, given a number of generation, determines how many of Willy's ancestors belong to that generation, under the assumption that the ancestors at each level are unrelated.

The Input

Your program receives a sequence of positive integers, one per line, each representing the generation. The maximum value for the generation is 80. The input terminates with a 0.

The Output

For each input case, your program must print the corresponding number of ancestors that Willy has in such generation.

Sample Input

```
1
2
3
0
```

Sample Output

```
1
2
3
```

Careful Teacher

Background

Once upon a time there was a teacher that wanted to teach his students all the words in the dictionary. To do so, he started selecting by memory two words of the dictionary, and wanted their students to be changing the leters of the words, one by one, so that the original word eventually got converted into the final word, but with the restriction that every step only a letter could be changed, and, very importantly, each intermediate word must also be part of the dictionary.

The Problem

You have to help the teacher to know beforehand if a word can be changed into another word, one letter at a time, and each intermediate word belonging to a given dictionary, so that the teacher can select interesting examples to his students.

The Input

Your program receives a list of (different) words, one each line, forming the dictionary, a line to end the dictionary, with the characters "`--`", and several lines, each containing two words separated by a space (starting word and ending word, respectively). For each pair, you have to tell if the first word can be converted into the second using the given dictionary. The dictionary will have fewer than 20000 words.

The Output

For each input word pair, your program must print "Yes" if the first word can be converted into the second, or "No" if it cannot be done.

Sample Input

```
abc
abd
acc
bbb
aba
--
abc acc
abc bbb
acc abd
bbb abc
abd abc
aba abc
```

Sample Output

```
Yes
No
Yes
No
```

Yes
Yes