2004 National Collegiate Programming Contest

- Problems: There are 8 problems (18 pages in all, not counting this cover page) in this packet.
- Problem Input: Input to the problem are done through the input files. Input filenames are given in the table below. The input file may contain one or more test cases. Test cases may be separated by any delimiter as specified in the problem statement.
- Problem Output: All output should be directed to standard output (screen output).
- Time Limit: The judges will run each submitted program with certain time limit (given in the table below).

Table 1: Problem Information Sheet

	Problem Name	Input File	Time Limit
Problem A	Three in a Row	pa.in	10 secs.
Problem B	GO	pb.in	10 secs.
Problem C	Throwing a Party	pc.in	10 secs.
Problem D	Leaf Comparison	pd.in	10 secs.
Problem E	Red and White	pe.in	10 secs.
Problem F	Polygon Covering	pf.in	10 secs.
Problem G	Subset Selection	pg.in	10 secs.
Problem H	Continued Fractions	ph.in	10 secs.

Problem A
Three-In-a-Row
Input File: *pa.in*

This game is called Three-In-a-Row (TIR). A game board with 4x4 tiles is shown below.

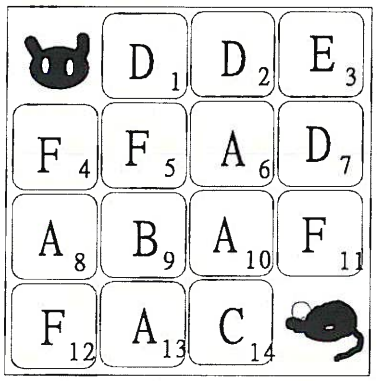


Figure 1: The TIR game board.

In the game board, there is a cat and a mouse (not necessarily placed in the corners). They are blocked by tiles. Each tile has an English letter on it. The rules of the game are as follows:

- If there are more than 3 (including 3) same tiles in a row (horizontal or vertical, **not** diagonal), they are destroyed immediately (i.e., removed from the game board). In Fig. 2, those tile patterns all satisfy this rule and are removed immediately.
- After no more tiles can be destroyed, a player can switch any two neighboring tiles to see if he/she can destroy any tiles which satisfy the 3-in-a-row rule. However, if no tiles can be destroyed, the switch is invalid. For example, in Fig. 1 you can not switch tile F_4 and A_8 because switching the two tiles does not destroy any tiles. However, switching B_9 and A_{13} is valid because it can destroy A_8, A_{13}, A_{10} which becomes 3-in-a-row.
- **Goal of the game:** to create a path between the cat and mouse to help the cat catch the mouse.

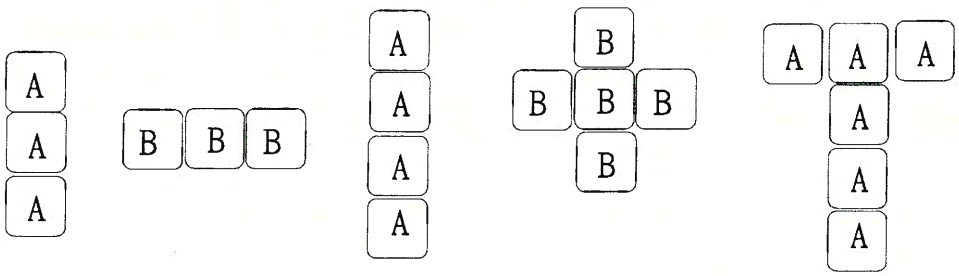


Figure 2: Some tile patterns which satisfy 3-in-a-row rule.

Let's look at a solution to create a path for Fig. 1. By switching E_3 and D_7, we can dig a path as in Fig. 3 (a). Next, by switching A_{13} and C_{14}, we can complete the path as in Fig. 3(b). On the other hand, from Fig. 3(a), if you mistakenly switch A_{13} and B_9 first, you enter a state with no further feasible steps to go on. Therefore, you fail to help the cat catch the mouse.

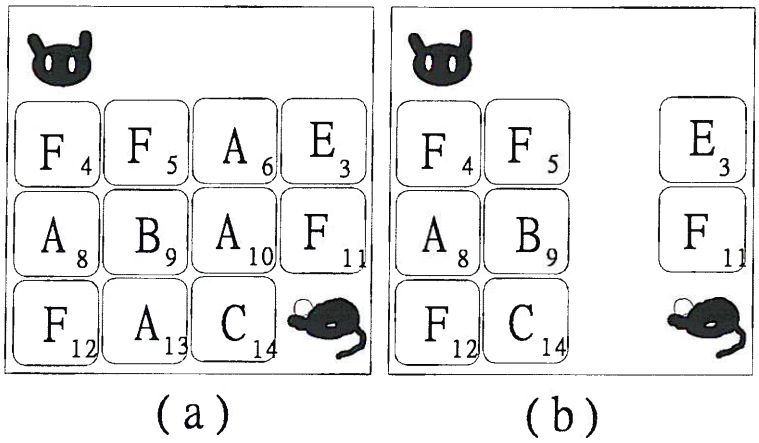


Figure 3: Digging a path from cat to mouse.

Given a TIR game, please write a program to answer if a game can allow the cat to catch the mouse.

Input File Format

The test file begins with a number N – the number of test cases. Each test case begins with two numbers (L L) to define the size of the game board, where 2 ¡= L ¡= 20. After (L L) is the data of tiles. "*" represents the cat. "@" represents the mouse. The other tiles are listed by capital English letters, as in sample input.

Output Format

For each test case, please print "yes" if the game allows the cat to catch the mouse. Please print "no" if otherwise.

Sample Input

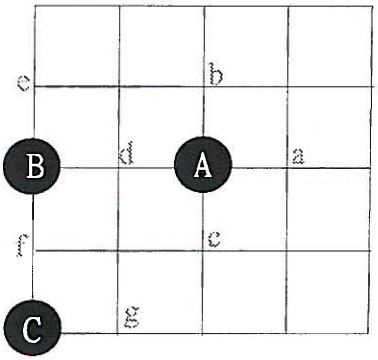
```
2
4~4~
{*}DDE
FFAD
ABAF
FAC@
4~4~
{*}AAC
DEFA
HIJC
@MNC
```

Output for the Sample Input

```
yes
no
```

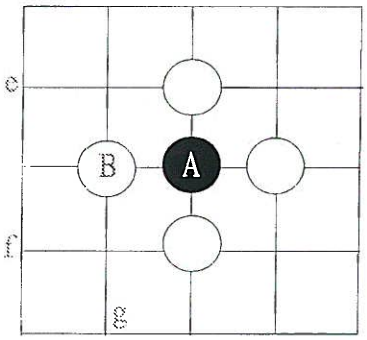
Problem B
GO
Input File: *pb.in*

GO is a well-known Chinese game. It is hard to play well. However, its rule is very simple. Consider the 5x5 game board below.



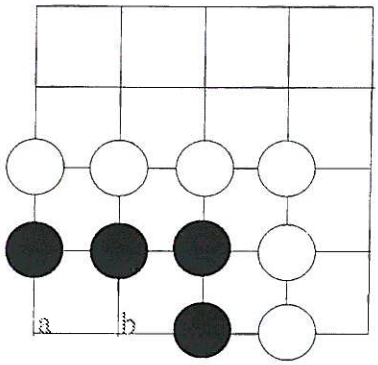
In the figure, there are three black stones placed on the game board. Stone A is placed in the middle of game board. It has 4 *airs* to breath. They are *a,b,c,* and *e*. Stone B, on the other hand, is placed along the edge of the game board. It has 3 airs to breath, which are *e,d,f*. Stone C is placed on the corner of the game board. It has 2 airs to breath, which are *f* and *g*.

To kill a stone in GO, you need to close the air of a stone. Consider the board below

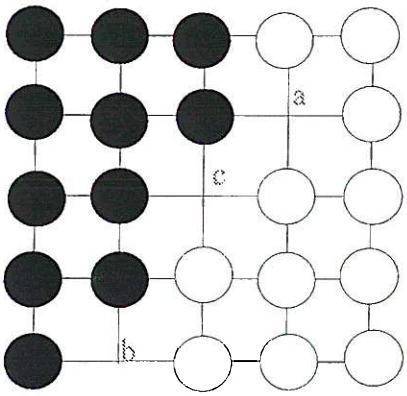


Black stone A is killed and removed from the board when white stone B is placed to close the last air of black stone A.

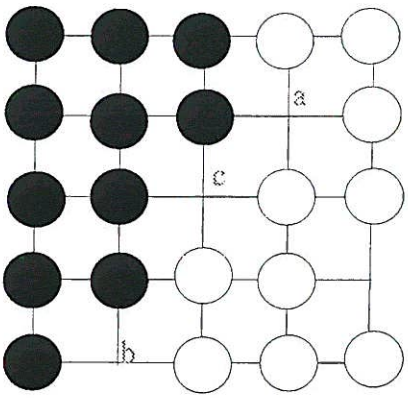
To kill a group of connected (connected in four directions) stones. Same rule applies. For example, in the game board below, the black stones have two airs, *a* and *b*. On the other hand, the white stones have 7 airs. So, the black stones are dead. White player can place a stone in *b* to close one air of black stones. Of course, black player can place a black stone at *a* to remove the white stone. However, white player can put another white stone at *b* again to close the last air of black stones and remove them all. The rule of GO is that you can put a stone in any place which has airs to breath. Or, if it has no airs to breath, the move can kill the opposite stones to create airs. Otherwise, putting a stone into a place which has no airs to breath is illegal.



So, in simple cases, determining the death or liveness of two groups of stones can be simply to compare the airs of each side. Ones with more airs kill the ones with less airs. If the number of airs is the same for both players, who has the next move wins. This simple rule, however, has some exceptions. One exception is shown below.



In this game, both black and white have 3 airs, *a, b,* and *c* but these airs are shared by both sides. In this case, the game is even. Any player attempts to make a move to kill the opposite kills himself. This is a situation called *double living.* No matter which player has the next move, no players win. On the other hand, if one player has one more free air, he wins. For example, in the following game, white wins. If both have same free airs and shared airs, you can conclude that the game is double living.



In this problem, your job is to write a program to determine which player in a 5x5 game wins **according to the rules and the exception described above.**

Note A: If you are familiar with GO, you do not need to consider other complicated exceptions in real GO game, such as double eyes. The test cases shall not test your program with cases which can not be determined by the rules described above. The death and liveness problem in real GO game is too complicated to deal with. Do not make this problem complicated if you are a real GO player.

Note B: You can assume the stones of a color are connected. That is, we will not test you with cases which contains two or more groups of disconnected black stones (or white stones). There is only one group of connected black stones and one group of connected white stones in test cases.

Input File Format

The input file begins with a number N - the number of test cases. In each test case, there are 5x5 characters to represent the game board as in sample input. A character can be ".", "B", or "W", where "B" and "W" are capital letters. "." represents no stones in the position. "B" represents a black stone and "W" represents a white stone. There is no space between letters. Letters are 5 in a row. After the data of 5x5 game board, a letter "B" or "W" is given to tell which player has the next move. Finally, behind the letter, "@" is given to signal the end of a test case.

Output Format

For each test case, if white player wins, please prints a "W". If black player wins, please prints a "B". If the test case is an even game, please print a "."

Sample Input

```
3
.....
.....
WWWW.
BBBW.
..BW.
B@
BBBWW
BBB.W
BB.WW
BBWWW
B.WWW
W@
BBBWW
BBB.W
BB.W.
BBWWW
B.WWW
B@
```

Output for the Sample Input

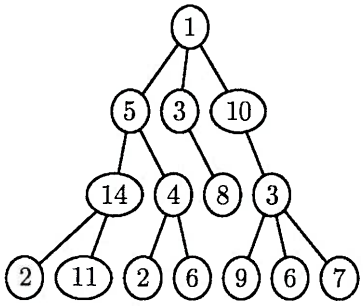
W

.

W

Problem C
Throwing a Party
Input File: pc.in

Consider a company that has a hierarchical structure; that is, the "supervisor" relation forms a tree rooted at the president. The personnel office has ranked each employee with a conviviality rating, which is an integer. An example is as follows.



You are responsible to plan a party for the company. In order to make the party fun for all participants, you do not want an employee and his/her immediate supervisor to attend at the same time. For instance, the two people with conviviality ratings 11 and 14 in the above example are not supposed to show up together, albeit their ratings are the highest in the company. The goal is to maximize the sum of the conviviality ratings of the guests. For instance, in the above example, a party with the highest sum (i.e., 66) of conviviality has to exclude the five people with ratings 1, 3, 4, and 14.

Technical Specification

1. There are n ($1 \leq n \leq 30$) people in the company, each of them has a unique ID from 1 to n. The ID of the president is 1.

2. For each $i = 1, 2, \ldots, n$, the conviviality rating r_i of the person with ID i is a positive integer no more than 30.

Input File Format

The first line of the input file contains an integer indicating the number of test cases to follow. Test cases are separated by a single blank line.

For each test case, the first line of input contains two integers n and r_1. For each $i = 2, 3, \ldots, n$, the i-the line of input contains two integers s_i and r_i, where s_i is the ID of the immediate supervisor of the person with ID i.

Output Format

For each test case, output the maximum sum of conviviality ratings of the guests.

Sample Input

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

16 1
1 5
1 3
1 10
2 14
2 4
3 8
4 3
5 2
5 11
6 2
6 6
9 9
9 6
9 7
```

Output for the Sample Input

```
15
66
```

Problem D
Leaf Comparison
Input File: *pd.in*

One of the methods for classifying hierarchical relations between different objects is by representing them in a tree. Different methods of classification may lead to different trees. In particular, trees have been used to represent evolutionary splits among species.

An n-node tree is a connected graph with $n-1$ edges. A *rooted tree* is a tree with a specified node called the *root*. For a node v in a rooted tree T, the *parent* of v, denoted by $par(v)$, is the first encountered node from v towards to the root. Node v is a *child* of $par(v)$. A node without children is called a *leaf*. Also let $T[v]$ denote the maximal subtree of T rooted at v. Figure 1(a) illustrates the structure of $T[v]$, and Figure 1(b) illustrates $T[2]$ of T.

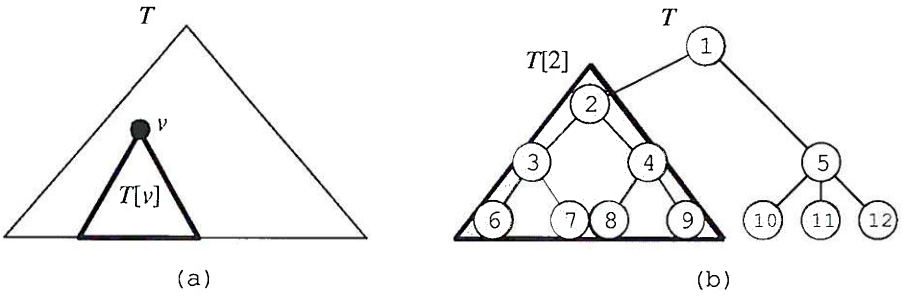


Figure 1: (a) The structure of $T[v]$. (b) $T[2]$ is shown within the bold triangle.

Suppose that there are two rooted trees T_1 and T_2 with N_1 nodes and N_2 nodes, respectively, such that the nodes of T_1 (respectively, T_2) are labelled from 1 to N_1 (respectively, 1 to N_2). Two *leaf-agree isomorphic* subtrees, denoted by $T_1[v_i] \cong T_2[v_j]$, where $v_i \in T_1$ and $v_j \in T_2$, are defined recursively as follows: Either v_i and v_j are two leaf nodes with the same label, or the children of v_i (say $\{v_{i_1}, v_{i_2}, ..., v_{i_m}\}$) and the children of v_j (say $\{v_{j_1}, v_{j_2}, ..., v_{j_m}\}$) can form a one-to-one corresponding such that $T_1[v_{i_1}] \cong T_2[v_{j_1}]$, $T_1[v_{i_2}] \cong T_2[v_{j_2}]$, ..., $T_1[v_{i_m}] \cong T_2[v_{j_m}]$.

Assume that T_1 has $2 \leq N_1 \leq 100$ nodes and T_2 has $2 \leq N_2 \leq 100$ nodes. Please write a program to count the cardinality of the *target set* $\{(v_1, v_2) |\ T_1[v_1] \cong T_2[v_2]$ for two non-leaf nodes $v_1 \in T_1$ and $v_2 \in T_2\}$.

For example, consider two rooted trees T_1 and T_2 shown in Figure 2(a) and (b), respectively. In this example, $T_1[2] \cong T_2[3]$, $T_1[4] \cong T_2[4]$, and $T_1[3] \cong T_2[5]$. Therefore, the output of this example equals 3 because the target set is $\{(2,3),(3,5),(4,4)\}$.

Input File Format The input consists of a number of test cases. Each test case consists of two rooted trees, which has the following format: The first line contains one positive integer $m_1 \leq 100$, which is the number of edges of the first tree. The next m_1 lines contain m_1 edges such that one line contains one edge. Each edge is represented by two positive numbers separated by a single space; the first number represents a node and the second one represents its parent. The (m_1+2)th line contains another positive integer $m_2 \leq 100$,

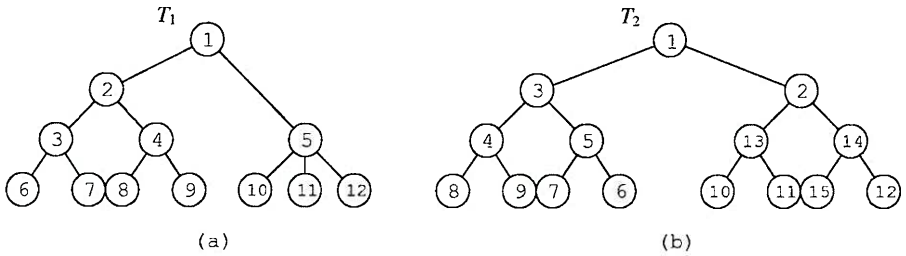


Figure 2: Two rooted trees T_1 and T_2 shown in (a) and (b), respectively.

which is the number of edges of the second tree. Then, the next m_2 lines contain m_2 edges of the second tree. Finally, a 0 at the $(m_1 + m_2 + 3)$th line indicates the end of the first test case.

The next test case starts after the previous ending symbol 0. A -1 singals the end of the whole inputs.

Output Format The output contains one line for each test case. Each line contains an integer, which is the cardinality of the target set.

Sample Input

```
11
6 3
7 3
8 4
9 4
10 5
11 5
12 5
3 2
4 2
2 1
5 1
14
8 4
9 4
7 5
6 5
10 13
11 13
15 14
12 14
4 3
5 3
13 2
14 2
3 1
2 1
```

```
0
1
1 2
1
2 1
-1
```

Output for the Sample Input

```
3
0
```

Problem E
Red and White
Input File: *pe.in*

In a box, there are balls colored either red or white. To discover the color of each ball, one must take the balls out of the box one at a time. If we know the numbers of white balls and red balls, then we might not have to remove all of the balls to find the color of each ball. The problem is to compute the average number of balls that one has to draw from the box for him/her to know the colors of all balls, providing that the numbers of red and white balls are known to be m and n respectively.

Technical Specification

1. There are n ($1 \leq n \leq 1000000$) red balls.
2. There are m ($1 \leq m \leq 1000000$) white balls.
3. The balls are drawn from the box one at a time, without replacement.

Input File Format

The first line of the input file contains an integer indicating the number of test cases to follow. Test cases are separated by a single blank line. Each test case contains two positive integers n and m denoting the number of red and white balls respectively.

Output Format

For each test case, compute the the average number of balls you have to draw from the box for you to know the colors of all balls. You have to output the number as a precise rational number $\frac{b}{a}$, where a and b are relatively prime positive integers. For each test case, you should output the pair (a, b).

Sample Input

```
2 3

7 15
```

Output for the Sample Input

```
2 7
16 315
```

Problem F
Polygon Covering
Input File: *pf.in*

A line segment L in the first quadrant of the xy-plane can be specified by four numbers a, b, p, and q such that for each point (x,y) on L, $0 \leq p \leq x \leq q$, and $y = a \times x + b \geq 0$. Let D_L be the polygon enclosed by the line segment L and the three lines x=p, x=q, and y=0. Given a set $S_L = \{h_1, h_2, \ldots, h_n\}$ of n numbers with $0 < h_1 < h_2 < \ldots < h_n$ and $\max(a \times p + b, a \times q + b) \leq h_n$, a polygon E_L is said to be a *covering* of D_L if each point in D_L is also in E_L and E_L can be decomposed into a finite number of rectangles through a set of vertical line segments such that the (vertical) height of each rectangle is an element in S_L. A covering of D_L is said to be *optimal* if its area is smallest among all coverings of D_L.

Given a line segment L specified by a, b, p, q, and a set $S_L = \{h_1, h_2, \ldots, h_n\}$, the problem asks to find the area of an optimal covering of D_L.

Example

Suppose L is specified by $a = 1, b = 2, p = 0, q = 10$, and S_L is $\{1, 5, 9, 13\}$. Figure 1 gives two coverings (whose areas are 90 and 100) of D_L, and the one on the left is optimal. Both coverings can be decomposed into three rectangles.

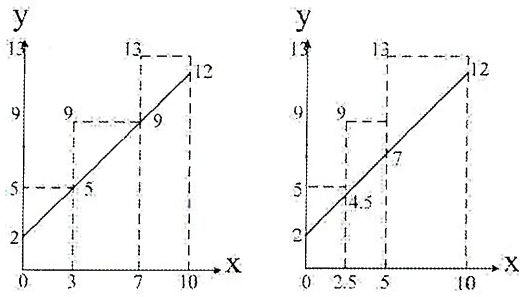


Figure 1: Two coverings.

Input File Format

The first line of the input file contains an integer indicating the number of test cases to follow. Test cases are separated by a single blank line.

For each test case, the first five lines give a, b, p, q, n, and the next n lines give $h_1, h_2, \ldots, h_n$. Note that $a, b, p, q, n, h_1, h_2, \ldots, h_n$ are all integers, and their ranges are: $-10 \leq a \leq 10$, $0 \leq b \leq 5000$, $0 \leq p < q \leq 5000$, $1 \leq n \leq 3500$, $1 \leq h_1 < h_2 < \ldots < h_n \leq 55000$, $0 \leq \min(a \times p + b, a \times q + b)$, and $\max(a \times p + b, a \times q + b) \leq h_n$.

Output Format

For each test case, print out the area of an optimal covering on a single line. Note that the area should be printed with at least 3 digits of precision to the right of the decimal point.

Sample Input

```
2
1
2
0
10
4
1
5
9
13

-2
111
0
47
8
13
48
74
83
91
115
156
164
```

Output for the Sample Input

```
90.000
3593.500
```

Problem G
Subset Selection
Input File: *pg.in*

Given an ordered set $X = \{x_1, x_2, \ldots, x_m\}$ of m non-decreasing integers, an ordered set $Y = \{y_1, y_2, \ldots, y_n\}$ of n non-decreasing integers and a positive integer k ($\leq n$), an ordered set $Z_Y = \{z_1, z_2, \ldots, z_k\}$ of k non-decreasing integers is said to be a *k-subset* of Y if each element in Z_Y is also in Y; besides, the cost of Z_Y is defined to be $\sum_{i=1}^{m} d(x_i, Q(x_i))$, where $d(x_i, Q(x_i))$ is the difference between x_i and $Q(x_i)$, i.e., $d(x_i, Q(x_i)) = |x_i - Q(x_i)|$. As for $Q(x_i)$, it denotes the element z_j in Z_Y which, among all elements in Z_Y, has the minimum difference from x_i, i.e., $Q(x_i) = z_j$ and $d(x_i, z_j) \leq d(x_i, z_l)$ for all $l \neq j$. Without loss of generality, if there is more than one element in Z_Y which has the same minimum difference from x_i, the one with the smallest index is chosen. A k-subset of Y is said to be *optimal* if its cost is smallest among all k-subsets of Y.

Given X, Y, and k, the problem asks to find the cost of an optimal k-subset of Y.

Example

Suppose $X = \{10, 20, 30, 40\}$, $Y = \{12, 24, 36\}$, and k=2. There are three 2-subsets of Y: {12, 24}, {12, 36}, and {24, 36}, and their costs are 28(=2+4+6+16), 20(=2+8+6+4), and 28(=14+4+6+4), respectively. Clearly, {12, 36} is optimal while the others are not.

Input File Format

The first line of the input file contains an integer indicating the number of test cases to follow. Test cases are separated by a single blank line.

For each test case, the first m+1 lines give m ($1 \leq m \leq 500$) and the m non-decreasing integers (each is between 0 and 35000) in X. The next n+1 lines give n ($1 \leq n \leq 500$) and the n non-decreasing integers (each is between 0 and 35000) in Y. The last line gives k.

Output Format

For each test case, print out the cost of an optimal k-subset on a single line.

Sample Input

```
2
4
10
20
30
40
3
12
24
```

```
36
2

10
28
59
66
80
92
94
115
120
121
122
5
29
30
54
88
106
3
```

Output for the Sample Input

```
20
115
```

Problem H
Continued Fractions
Input File: *ph.in*

Let $a_0, a_1, a_2 \cdots, a_n$ be integers with $a_i > 0$ for $i > 0$. A continued fraction(CF) of order n with coefficients $a_1, a_2 \cdots, a_n$ and the initial term a_0 is defined by the following expression

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \cdots + \frac{1}{a_n}}}$$

which can be abbreviated as $[a_0; a_1, a_2, \cdots, a_n]$.

An example of a CF of order 3 is $[1; 2, 3, 4]$, which is equivalent to

$$1 + \frac{1}{2 + \frac{1}{3 + \frac{1}{4}}} = \frac{43}{30}$$

We can see that any CF is rational, since it can be expressed as a fraction. Write a program that converts any CF to a fraction in lowest terms, that is, the numerator of the fraction is relative prime to its denominator.

Input File Format

The input contains several of problem sets, one set in a line. Each problem set(line) consists of $n + 2$ numbers, which are $n (1 \leq n \leq 12)$ the order of the CF and followed by $n + 1$ numbers for $a_0, a_1, \cdots, a_n$ in sequence. Each value of a_i for $i \geq 0$ is between 1 and 5. The last set will consist of a single number 0 with no other number, which signifies the end of input.

Output Format

Your program should print one line for each problem set, where the numerator is printed first then followed by the denominator.

Sample Input

```
3 1 2 3 4
3 2 1 4 3
2 2 2 1
0
```

Output for the Sample Input

```
43 30
45 16
7 3
```