Clarification for 2019 District
COMPUTER SCIENCE - PROGRAMMING

COMPUTER SCIENCE - PROGRAMMING

For **Problem # 7 (Katya),** there is a discrepancy between the sample input in the printed student problem set and the student data file `katya.dat`. The data file is correct. The corresponding judge data file and printed judge packet are unaffected. *No modifications to any files are necessary.*

Contest directors should announce to teams at the beginning of the contest period that this discrepancy exists for Problem 7, and that they should work only from their sample data file and ignore the Sample Input in the printed problem statement.

(It's likely that most teams will be working from their sample data files anyway and would not notice the discrepancy, but we want to ensure there is no confusion.)

Computer Science Competition
District 2019
Programming Problem Set

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.

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.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Bipul
Problem 2	Banu
Problem 3	Botan
Problem 4	Dawn
Problem 5	Egor
Problem 6	Gowri
Problem 7	Katya
Problem 8	Luciano
Problem 9	Mateo
Problem 10	Rajeev
Problem 11	Sarah
Problem 12	Tammy

1. Bipul

Program Name: Bipul.java Input File: None

Nepal is the only nation in the world that does not have a quadrilateral flag! The flag instead looks like two overlaid pennants (a kind of triangle). The pennants have different shapes and sizes, which makes reproducing the Nepalese flag faithfully a difficult task.

Bipul wants to make an ASCII-art outline of a Nepalese flag. In his recreation, both pennants have a width of 11 characters. Furthermore, the top pennant overlaps the tip of the bottom pennant.

Input: There is no input file for this problem.

Output: Output the Nepalese flag as shown below.

Expected Exact Output:

```
|
|#\
|###\
|#####\
|#######\
|#########\
|##\
|###\
|####\
|#####\
|######\
|#######\
|########\
|#########\
```

2. Banu

Program Name: Banu.java Input File: banu.dat

Banu is a very busy woman! She has made several appointments to meet with friends, family and coworkers and entered them into a text file. For each appointment she has listed the name followed by the date. However, she has not entered the dates in the format she prefers and would like to change it.

Being very new to programming and processing data, she decides to begin her conversion program with just the month number, changing it to the appropriate month name. For example, the month number 1 would be changed to January, 2 to February, and so on.

Input: Several integers in the range 1 – 12, inclusive, each on one line, each representing one of the twelve months of the year.

Output: The corresponding month name for the month number, output in initial upper-case format, aka title format, as shown below.

Sample input:

```
5
2
7
12
```

Sample output:

```
May
February
July
December
```

3. Botan

Program Name: Botan.java Input File: botan.dat

Poor Botan is very confused. He just learned about prime numbers, numbers with only two positive integer divisors. The first few prime numbers are 2, 3, 5, 7, 11, and 13. It's an active area of research to discover large prime numbers, as they have applications in cryptography. Botan has a solution to this problem: just change the definition of what a prime number is and call it "Botan-prime"!

To convince people to go along with the "Botan-prime", a number less than 10 is a Botan-prime if and only if it is actually prime. For numbers greater than or equal to 10, a number is "Botan-prime" if and only if all of its substrings are "Botan-prime".

A substring of a number is a continuous block of digits from that number. For example, 3892 has 9 substrings: 3, 38, 389, 3892, 8, 89, 892, 92, and 2. Those are the only substrings of 3892. A proper substring is a substring that is not the original. Thus 3892 has 8 proper substrings: 3, 38, 389, 8, 89, 892, 92, and 2.

32 is not a substring of 3892 because the digits are not consecutive. 98 is not a substring of 3892 because those digits do not appear in that order in the number. This number is not Botan-prime, as 8 is not Botan-prime.

Given a number, can you tell if it is Botan-prime or not?

Input: The first line is positive integer T (at most 20), the number of test cases to follow. Each test case contains a single positive integer N (at most 1,000,000).

Output: For each test case, print the test case number and whether the input is Botan-prime or not. Follow the formatting in the samples.

Sample input:

```
5
3
1
15
35
3892
```

Sample output:

```
Case #1: BOTAN-PRIME
Case #2: NOT BOTAN-PRIME
Case #3: NOT BOTAN-PRIME
Case #4: BOTAN-PRIME
Case #5: NOT BOTAN-PRIME
```

4. Dawn

Program Name: Dawn.java Test Input File: dawn.dat

Dawn has seen news stories about people winning big lottery drawings with the option to receive either a single lump sum payment or a series of payments over several years from an annuity. She wondered how such an annuity would work if she were to receive a large inheritance from a long-lost relative, "Uh sure," she thought to herself. But, she decided this would be an opportunity to use her programming skills.

During her research, she discovered several options for annuity payments and settled on the following formula, calculating a fixed monthly payment for an annuity with a fixed annual percentage rate (APR) and no further deposits toward the initial investment.

Formula	Legend
$$p = \frac{ar}{1-(1+r)^{-n}}$$	p = amount of payment a = amount invested r = monthly interest rate n = number of payments

The investment amount is standard currency with dollars and cents and the amount could reach $250,000.00. The monthly interest rate is one-twelfth the APR as a decimal value with the APR provided as a percentage like 5.995%. The length of the annuity is the number of years, a whole number, for which Dawn will receive monthly payments.

Dawn is aware of floating-point issues in programming and recalls her computer science teacher telling them when working with money, calculated amounts must be rounded to the nearest $0.01 before using them in subsequent computations; otherwise, partial pennies eventually turn into whole pennies causing later computations to produce incorrect values. In order to test her handling of money values, Dawn decides she will produce some totals to check her rounding.

Input: The first line of the data file contains a count of the number of data sets, with a max of 20 to prevent totals from exceeding the field widths shown below. Each set of data will contain a line with three floating-point values: amount invested, annual percentage rate (APR), and number of years. Whitespace separates all values.

Output: A table containing the following items, formatted exactly as shown, **with alignment dots** in the second line of column headers and vertical lines "|" between columns. The last line contains totals for those columns. Input values and calculated results will not exceed the field widths shown below.

```
     Amount                              Monthly          Total of
....Invested........APR......Years......Payment.........Payments............Profit..
$  130,360.20  |    6.825%  |    36  |  $    811.44  |  $  350,542.08  |  $  220,181.88
$  131,548.95  |  10.020%   |    25  |  $ 1,197.24  |  $  359,172.00  |  $  227,623.05
$  154,472.61  |    3.657%  |    12  |  $ 1,326.87  |  $  191,069.28  |  $   36,596.67
$  116,466.55  |    3.478%  |    39  |  $    454.98  |  $  212,930.64  |  $   96,464.09
$  532,848.31  |            |        |  $ 3,790.53  |  $1,113,714.00  |  $  580,865.69
```

Sample input:

```
4
130360.20    6.825      36
131548.95   10.020      25
154472.61    3.657      12
116466.55    3.478      39
```

Sample output: The exact table with precise formatting as shown above but without the outside border lines.

5. Egor

Program Name: Egor.java Test Input File: egor.dat

Egor and his wife Zelda are going to have a baby. Congratulations! The big day is approaching quickly, and the happy couple have made all the necessary preparations. They have purchased a car seat, set up a crib and even painted the nursery. But they have been procrastinating on one very important step in their preparations. They haven't chosen a name yet! Turns out they both have some very odd requirements for the name of their baby.

Egor has never been very fond of his name so he would rather not name the baby Egor. He does, however, like that his name is short and easy to spell so he would like to limit the number of letters in his baby's name to eight, so a name like Bernadette would be out. Furthermore, Egor thinks that the "ph" sound is a nuisance so names like Stephen and Ralph are out.

Zelda, on the other hand, can't stand the letters c, m, o and y. So that's a hard no on Mary and Coy. She thinks very short names are not sophisticated enough so she wants her child to have at least five letters in its name. Her child will not be named Ty. Zelda doesn't like that her name comes at the end of the alphabet so her last requirement for her baby's name is it must begin with a letter from the first half of the alphabet.

All these requirements make it really difficult to choose a name! These guys could use a program to help them sift through possible names and come up with a list of names that meets all requirements. Help them out by writing that program.

Input: The first line of data will be a number N that represents the number of possible names, followed by N more lines each containing one possible name. Each name will always begin with a capital letter and will contain no symbols. There will be no duplicate names.

Output: An alphabetized list of acceptable names, each on a separate line.

Sample input:

```
16
Bipul
Banu
Dawn
Egor
Gowri
John
Kirby
Richard
Joel
Arnav
Carlanthony
Danica
Alok
Inisha
Bernadette
Calvin
```

Sample output:

```
Arnav
Bipul
Inisha
```

6. Gowri

Program Name: Gowri.java Test Input File: gowri.dat

Gowri loves shapes and needs a program that makes a weird shape. Gowrie does not like normal shapes and wants a square, but with the corners angled with three asterisks. For instance, this is a 4 X 4 square, with the 3-asterisk diagonal corners.

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

Input: The first line is T (T ≤ 100), the number of data sets. Each data set has one integer >1 and ≤ 20.

Output: A square made up of asterisks with corner diagonals of exactly 3 asterisks, each pattern followed by a row of 20 dashes.

Example Input:

```
2
4
2
```

Example Output:

```
     ****
    *    *
   *      *
  *        *
 *          *
*            *
*            *
*            *
*            *
 *          *
  *        *
   *      *
    ****
--------------------
    **
   *  *
  *    *
 *      *
*        *
*        *
 *      *
  *    *
   *  *
    **
--------------------
```

7. Katya

Program Name: Katya.java **Input File: katya.dat**

Katya saw her brother working with 2-dimension arrays for his college math homework. He explained that the arrays are called matrices in math and he was performing matrix multiplication. He demonstrated the following example to his sister.

Start with two matrices (2-dimension arrays), matrix **A** with **n** rows and **m** columns and matrix **B** with **m** rows and **p** columns. The number of columns **m** in **A** must equal the number of rows **m** in **B**. Their product **C = AB** is a matrix **C** with **n** rows and **p** columns. Position **(i,j)** in the result matrix **C** is the sum of **m** products, the elements in row **i** of **A** multiplied by the elements in column **j** of **B**.

Matrix A n × m 3 × 4	Col 1	Col 2	Col 3	Col 4
Row 1	1	0	-3	2
Row 2	2	-3	0	2
Row 3	-3	0	2	3

Matrix B m × p 4 × 2	Col 1	Col 2
Row 1	-1	0
Row 2	2	-4
Row 3	-3	3
Row 4	0	-2

Matrix C n × p 3 × 2	Col 1	Col 2
Row 1	8	-13
Row 2	-8	8
Row 3	-3	0

C(1,1) sums the products of row 1 from A and col 1 from B
= A(1,1) × B(1,1) + A(1,2) × B(2,1) + A(1,3) × B(3,1) + A(1,4) × B(4,1)
= 1 × -1 + 0 × 2 + -3 × -3 + 2 × 0 = 8
C(1,2) sums the products of row 1 from A and col 2 from B
= A(1,1) × B(1,2) + A(1,2) × B(2,2) + A(1,3) × B(3,2) + A(1,4) × B(4,2)
= 1 × 0 + 0 × -4 + -3 × 3 + 2 × -2 = -13
C(2,1) sums the products of row 2 from A and col 1 from B
= A(2,1) × B(1,1) + A(2,2) × B(2,1) + A(2,3) × B(3,1) + A(2,4) × B(4,1)
= 2 × -1 + -3 × 2 + 0 × -3 + 2 × 0 = -8
C(2,2) sums the products of row 2 from A and col 2 from B
= A(2,1) × B(1,2) + A(2,2) × B(2,2) + A(2,3) × B(3,2) + A(2,4) × B(4,2)
= 2 × 0 + -3 × -4 + 0 × 3 + 2 × -2 = 8
C(3,1) sums the products of row 3 from A and col 1 from B
= A(3,1) × B(1,1) + A(3,2) × B(2,1) + A(3,3) × B(3,1) + A(3,4) × B(4,1)
= -3 × -1 + 0 × 2 + 2 × -3 + 3 × 0 = -3
C(3,2) sums the products of row 3 from A and col 2 from B
= A(3,1) × B(1,2) + A(3,2) × B(2,2) + A(3,3) × B(3,2) + A(3,4) × B(4,2)
= -3 × 0 + 0 × -4 + 2 × 3 + 3 × -2 = 0

Katya realizes that is a lot of work for even a small example and decides to write a program to help her brother check his calculations.

Input: The first line of the data file contains the number of data sets. Each data set will contain a line with four integers: the number of rows r_1 and columns c_1 in the first array and the number of rows r_2 and columns c_2 in the second array. That line will be followed by r_1 lines each contains values for c_1 columns and then r_2 lines each contains values for c_2 columns. Whitespace separates all values. Maximum matrix size is 10 × 10.

Output: For each data set, display a line that starts with the data set number followed by a colon. If array sizes are compatible ($c_1 = r_2$) display the number of rows and columns of the resulting array after the colon with the numbers separated by a comma. If not compatible, display the message "**SIZES NOT COMPATIBLE**" after the colon with no extra spacing. When they are compatible, display one line for each row of the resulting array with the column values displayed right aligned in fields exactly 5 characters wide and no additional spacing. After each data set, output a line containing exactly 12 carats "**^^^^^^^^^^^^**".

Katya (cont.)

Sample input:

```
3
3  4  4  2
 1  0 -3  2
 2 -3  0  2
-3  0  2  3
-1  0
 2 -4
-3  3
 0 -2
4  6  5  4
 7  6 -7  3 -1  4
-4  8 -1  8 -3 -8
 2 -1 -6 -6  1 -6
 4  5  1  6 -7  4
 4  7 -9 -3
 6 -9 -1 -4
-1  4 -5  6
 9 -9 -3  0
 8  6 -3  5
4  3  3  5
 4 -1  3
 3 -1  1
 0  7 -1
 6  2  7
-1  6  2 -5  3
 3 -8  1 -1  3
-4 -6 -7  3  4
```

Sample output:

```
1:3,2
    8  -13
   -8    8
   -3    0
^^^^^^^^^^^^
2:SIZES NOT COMPATIBLE
^^^^^^^^^^^^
3:4,5
  -19   14  -14  -10   21
  -10   20   -2  -11   10
   25  -50   14  -10   17
  -28  -22  -35  -11   52
^^^^^^^^^^^^
```

8. Luciano

Program Name: Luciano.java **Test Input File: luciano.dat**

Luciano is learning sorts and has discovered different ones have different behaviors. He has a particular fascination with the **bubble** and **selection** sorts and wants to compare the two. Since final results are the same for both (the entire list in sorted order) and do not provide any obvious differentiation between the processes, he decides to stop each process about halfway through and study those results. Luciano chooses to sort the items in ascending alphabetical order, and only works with lowercase words.

For example, using a short list containing the words:

```
one two horse house habitat
```

the selection sort starts at the beginning, seeking the best word for the first place in the list (`habitat`), swapping it with the current first word in the list (`one`), and then for the second place (`horse`), swapping it with the current second word (`two`), and so on. After two iterations the order of the list is this, with the first two words in correct places:

habitat horse two house one

In the standard bubble sort, the word last in alpha order (`two`) is "bubbled" to the back in the first iteration, then the next to last in order (`one`), to the penultimate place, and so on, ending with this order after two iterations, the last two in correct places, and the remaining words in the same original relative order:

horse house habitat **one two**

When he realizes the bubble sort works from the end of the list, and the selection sort from the beginning, he decides to alter the bubble sort to work from the beginning to make it easier to study the difference in the behavior of the two sorts for the unprocessed portions of the list. This **reversed bubble sort** would "bubble" the first words in alpha order to the front of the list, resulting in the first two words in the correct place, showing this order after two iterations:

habitat horse one two house

Now he can see and study the slight difference in the unsorted portions of each list, satisfying his research goal.

Input: An integer N ($20 \leq N \leq 40$) stating the number of items in the dataset, followed by N words to be sorted, each of length 20 characters or less, all in lowercase.

Output: Two columns, the showing the selection sort order after **the first 10 items are sorted**, the second column the bubble sort order, also after **the first 10 items are sorted**. The columns will be separated by a colon (:). Each column will be 20 characters wide, left aligned, with a column header accordingly. See the output example for the exact format to follow.

See sample input and output for Luciano on next page.

Luciano (cont.)

Sample Input: **Sample Output:**

20	Selection Sort	:Bubble Sort
one	aaron	:aaron
two	bigilow	:bigilow
horse	blue	:blue
house	canary	:canary
habitat	canton	:canton
hunday	crown	:crown
gmc	detroit	:detroit
aaron	eightenn	:eightenn
uilproblems	eleven	:eleven
hairstudio	gmc	:gmc
eleven	uilproblems	:one
hairdo	hairdo	:two
canary	house	:horse
canton	habitat	:house
crown	hunday	:habitat
bigilow	two	:hunday
eightenn	one	:hairdo
blue	horse	:hairstudio
detroit	hairstudio	:uilproblems
hillary	hillary	:hillary

9. Mateo

Program Name: Mateo.java Test Input File: mateo.dat

Mateo has been studying polynomial functions in his algebra class and has no problem using the standard quadratic formula to find the roots, (or lack thereof) for a quadratic function. Higher-level polynomial functions are another matter when it comes to finding the roots because there is no simple formula. Mateo's computer science teacher explained that computational techniques could sometimes be used to find roots.

His teacher explained that if the sign of the function value changes from positive to negative or from negative to positive for two values of x, there is at least one root in that interval where $f(x) = 0$. However, when two values of x result in $f(x)$ being either positive or negative for both x values, it is unknown whether a root exists between those two values. When the values of $f(x)$ have opposite signs for a pair of x values, Mateo plans to select a new value for x between the two given values and then use that new value to shrink the interval in which the root exists until he closes in on a value where $f(x) = 0$.

Mateo decides to write a program to try the technique using the following polynomial function:

$$f(x) = 1.3x^4 - 35.1x^2 - 18.2x + 89.7$$

Assume at most one root will exist between each pair of x values. Realizing accuracy of floating-point values is an issue in programming, Mateo decides to use $f(x) = 0.0 \pm 0.0000001$ to find the root.

Input: An unknown number of lines and each line contains two different floating-point values for x, separated by whitespace, neither a root.

Output: When the pair of x values result in no sign change in $f(x)$, display "UNKNOWN"; otherwise, display the value of the root's x value with seven decimal places of accuracy.

Sample input:

```
-5.0 -3.0
0.0 -5.0
1.0 -4.0
-3.5 3.5
```

Sample output:

```
-4.5337926
UNKNOWN
-2.0804001
UNKNOWN
```

10. Rajeev

Program Name: Rajeev.java **Test Input File: rajeev.dat**

Rajeev works for UIL and uses Google, which allows users to share information with other UIL team members. Rajeev has now discovered that some of the confidential UIL files are being shared with people outside of UIL, creating a potentially insecure environment.

The good news is that Google allows administrators to display all files that are shared within an organization. This is called the "Exposure File Report." There is a lot of data, and Rajeev only cares about the files that are shared outside of the organization.

Your job is to create a quick program that displays all the files and accounts with access to that file when at least one account with access is not a UIL account. Rajeev has discovered the format Google uses to display a file is: the file name, followed by a comma, then the owner of the file, then a comma, then a comma-separated list of email addresses that have access to the file. UIL has two different domains (email address endings). All UIL email addresses end with @uiltexas.org or @uil.net.

Example: For the two sample input lines below, the first line shows a file only shared with UIL accounts, so it does not need to be printed. The second file is shared with a **yahoo.com** account, so it should be printed.

```
Anubhav.java, director@uil.net, hammon@uiltexas.org, trussel@uil.net
Anubhav.doc, director@uil.net, hammon@uiltexas.org, otheremail@yahoo.com,
trussel@uil.net
```

Input: Each line contains the file name followed by the owner, followed by every email address with which the file is shared, separated by commas.

Output: For each line containing an email address not associated with the UIL, display the file name separated by a colon, followed by the original owner, another colon, and then followed by all invalid email addresses, separated by colons. Output should be displayed in the same order as the input.

Example Input:

```
file1.txt,sanchez@uiltexas.org,corley@uiltexas.org,guerrero@uiltexas.org,narvaez@uiltexas.org,campbell@uil.net
file2.org,cameron@uiltexas.org,rodriguez@uil.net,phillips@uiltexas.org,sanchez1@uiltexas.org,moore@uiltexas.org,benitez@uiltexas.org
smiley.jpg,wawarofsky@uiltexas.org,romero@uiltexas.org,trujillo@uiltexas.org,turrubiart@uil.net,villa@uiltexas.org,martinez@uiltexas.org
aaron.doc,henry@uiltexas.org,arellano@uiltexas.org,perez@yahoo.com,vargas@uiltexas.org,benningfield@uiltexas.org
jacob.jpg,calixto@uil.net,jaimes@uiltexas.org,richardson@uiltexas.org,villa@uiltexas.org,dozier@uiltexas.org
cobra.gif,hughes@uiltexas.org,medina@uiltexas.org,hernandez@uil.net,mcfaden@uiltexas.org
contra.gsheet,woolley@uiltexas.org,camacho@uiltexas.org,peterson@uiltexas.org,perez@uil.com,dozier@uiltexas.org,nomura@uiltexas.net
one.csv,yarbrough@uiltexas.org,sanchez@uiltexas.org,palafox@uiltexas.org,peterson@uiltexas.org
two.csv,corley@uiltexas.org,torres@uiltexas.org
```

Example Output:

```
aaron.doc:henry@uiltexas.org:perez@yahoo.com
contra.gsheet:woolley@uiltexas.org:perez@uil.com:nomura@uiltexas.net
```

11. Sarah

Program Name: Sarah.java Input File: sarah.dat

Sarah is making a word scramble game. To scramble the words to be used in her game, Sarah switches a couple of the letters around each of the vowels in the word. An example would be that ***computer*** is scrambled to be ***mocturep***. Sarah has come up with three rules for how the words are to be scrambled. They are:

Working from left to right in each word:

1. If the first letter is a vowel, switch the first and second letter.
2. After the first letter, each time a vowel is encountered, switch the letters that come immediately before and after the vowel.
3. If the last letter is a vowel switch it with the first letter in the word.

Using the word **computer**, the first change follows Rule 2, switching the letters before and after the "o", resulting in **mocputer**. When the "u" is reached, the "p" and "t" are switched, yielding the new word "**moctuper**". At the "e", another switch is made, "**mocpuret**" as the final result.

Sarah is much better at designing games than she is at writing code, so she needs your team's help. Write a program that will take a variety of words and scramble them using Sarah's flipping algorithm.

Input: An unknown number of words each separated by exactly one space with several words on each of several lines. All words will be entirely in lower case letters.

Output: For each word, in the order they are encountered in the file, print the original word followed by exactly one space and then the scrambled word.

Sample input:

```
computer science rocks
texas is the second
largest state in the
united states by both
area and population
```

Sample output:

```
computer mocturep
science eeicncs
rocks corks
texas xesat
is is
the eht
second cenosd
largest ralsegt
state etats
in in
the eht
united iundet
states staset
by by
both tobh
area aare
and dan
population polutaoipn
```

12. Tammy

Program Name: Tammy.java **Test Input File: tammy.dat**

Tammy is learning about graphs! A *graph* is a set of vertices and a set of edges, where each edge connects two vertices. A *path* is a sequence of vertices, v1, v2, ..., vn where there is an edge between every two consecutive vertices. A *simple cycle* is a path that begins and ends at the same vertex, and doesn't repeat any edges.

A *coloring* of a graph is an assignment of colors to vertices such that every edge in the graph connects vertices of different colors.

You are given a graph with **no simple cycles**. Find the minimum number of colors needed to color the graph, and the number of ways to color the graph with that many colors. Two colorings of a graph are considered different if at least one vertex has a different color in the colorings.

For example, the first sample input below requires two colors. Say the colors are black and white. Then these are the possible colorings:

The second sample only has one node, so it only requires one color, with only one way to color it!

Input: The first line is T (T <= 10), the number of test cases to follow. The first line of each test case is two space-separated integers V and E, the number of vertices and number of edges respectively. The following E lines each contain two space separated integers a and b, meaning there is an edge between vertices a and b. It is guaranteed each edge connects two different vertices and there is at most one edge between any pair of vertices. It is also guaranteed that this graph has no simple cycles.

1 <= V <= 50
0 <= E < V
1 <= a, b <= V

Output: For each test case, print two integers C and W separated by a space. C is the minimum number of colors needed to color the graph, and W is the number of ways to color the graph.

Sample input:

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

Sample output:

```
2 2
1 1
```