Maratón de Programación 2020

12/12

Java

python

C/C++

“Triunfar en la vida no es ganar, es levantarse y volver a empezar cada vez que uno cae”

(Pepe Mujica)

Universidad Francisco de Paula Santander

Vigilada Mineducación

Programa de Ingeniería de Sistemas

Acreditado de Alta Calidad

“Educación y Tecnología con Compromiso Social”

GitHub

Contents

1 Maratón de Programación UFPS 2020

Desde el año 2015, el programa Ingeniería de Sistemas de la Universidad Francisco de Paula Santander (UFPS) inició un interesante proceso para promover la Programación Competitiva, como parte de las actividades del Semillero SILUX (Linux, Software Libre y Licencias Abiertas). El propósito fundamental fue el desarrollo de competencias de resolución de problemas, programación de computadores, trabajo colaborativo y liderazgo.

Los pioneros de dicho proyecto fueron dos estudiantes, quienes ya se graduaron y dejaron como herencia una gran motivación. Ahora siguen colaborando con el Semillero que es liderado directamente por los estudiantes, quienes ingresan desde primer semestre con el entusiasmo que trae el sueño de llegar algún día a la Competencia Mundial ICPC (The International Collegiate Programming Contest https://icpc.baylor.edu/).

Desde entonces, cada año, el evento más importante es la Maratón Interna de Programación de la UFPS, que propone un conjunto de retos para poner a prueba las habilidades en algoritmia, programación y trabajo en grupo de los estudiantes. En dicho evento un actor fundamental siempre ha sido la Red de Programación Competitiva (RPC), quien apoya toda la logística de preparación de la competencia y además ofrece su plataforma tecnológica y su equipo de trabajo. El lema de RPC es "Aquí crecemos juntos" y la UFPS ya lleva seis (6) años creciendo en Programación Competitiva junto a muchas universidades en varios países de toda Latinoamerica.

Para éste año 2020 nos complace presentar un conjunto de quince (15) problemas inéditos, los cuales fueron escritos por estudiantes, procesores y graduados de varias universidades. Por la pandemia del COVID19 este año nuestra maratón es virtual, con lo cual damos un ejemplo de confianza, transparencia y visión de lo que será el futuro de estas competencias.

Reconocimiento y agradecimiento especial al equipo de trabajo:

- José Manuel Salazar Meza - Estudiante UFPS (desde Cúcuta)
- Juan Camilo Bages - Graduado Uniandes Bogotá (desde USA)
- Crysel Jazmin Ayala Llanez - Estudiante UFPS (desde Cúcuta)
- Carlos Fernando Calderón Rivero - Estudiante UFPS (desde Cúcuta)
- Milton Jesús Vera Contreras - Profesor UFPS (desde Cúcuta)
- Gerson Yesid Lázaro - Graduado UFPS (desde Bogotá)
- Angie Melissa Delgado - Graduado UFPS (desde Cali)
- Hugo Humberto Morales - UTP Pereira
- Equipo Red de Programación Competitiva (Fabio Avellaneda y Jovani Careño)

2 Grupo de Estudio en Programación Competitiva

El grupo de estudio en Programación Competitiva hace parte del Semillero de Investigación SILUX (Linux, Software Libre y Licencias Abiertas) y tiene como fin preparar y fortalecer a los estudiantes de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander en competencias de resolución de problemas, algoritmia, programación de computadores, trabajo colaborativo y liderazgo. Se aprovecha el entorno competitivo o gamificación porque favorece el aprendizaje tanto de habiidades hard como habilidades soft.

Los integrantes del grupo de estudio han participado en la Maratón Nacional de Programación desde el año 2014, logrando por 6 años consecutivos la clasificación a fase Regional Latinoamericana.

3 Instrucciones

Puedes utilizar Java, C, C++ o Python, teniendo en cuenta:

1. Resuelve cada problema en un único archivo. Debes enviar a la plataforma únicamente el archivo .java, .c, .cpp, o .py que contiene la solución.

2. Todas las entradas y salidas deben ser leídas y escritas mediante la entrada estándar (Java: Scanner o BufferedReader) y salida estándar (Java: System.out o PrintWriter).

3. En java, el archivo con la solución debe llamarse tal como se indica en la linea "Source file name" que aparece debajo del título de cada ejercicio. En los otros lenguajes es opcional (puede tener cualquier nombre).

4. En java, la clase principal debe llamarse igual al archivo (Si el Source File Name indica que el archivo debe llamarse example.java, la clase principal debe llamarse example).

5. En java, asegúrate de borrar la linea "package" antes de enviar.

6. Tu código debe leer la entrada tal cual se indica en el problema, e imprimir las respuestas de la misma forma. No imprimas lineas adicionales del tipo "La respuesta es..." si el problema no lo solicita explicitamente.

7. Si su solución es correcta, en unos momentos la plataforma se lo indicará con el texto "YES". En caso contrario, obtendrá un mensaje de error.

4 Reglas

1. Gana el equipo que resuelve mas problemas. Entre dos equipos que resuelvan el mismo número de problemas, gana el que los haya resuelto en menos tiempo (ver numeral 2).

2. El tiempo obtenido por un equipo es la suma de los tiempos transcurrido en minutos desde el inicio de la competencia hasta el momento en que se envió cada solución correcta. Habrá una penalización de 20 minutos que se suman al tiempo por cada envío erróneo (esta penalización solo se cuenta si al final el problema fue resuelto).

3. NO se permite el uso de internet durante la competencia. Únicamente se puede acceder a la plataforma en la cual se lleva a cabo la competencia.

4. NO se permite el uso de dispositivos electrónicos durante la competencia (excepto el computador usado para competir).
5. NO se permite la comunicación entre miembros de equipos diferentes. Cada integrante solo puede comunicarse con sus dos compañeros de equipo.
6. Se permite todo tipo de material impreso (libros, fotocopias, apuntes, cuadernos, guías) que el equipo desee utilizar.
7. NO se permite copiar y pegar código desde fuentes disponibles en Internet.

5 Lista de Problemas

A continuación la lista de los quince (15) problemas a resolver. Un primer reto y logro es resolver alguno de los problemas. Un segundo reto es lograr resolver cualquier de los problemas más rápido que todos los demás. Y un tercer reto es lograr resolver todos los problemas.

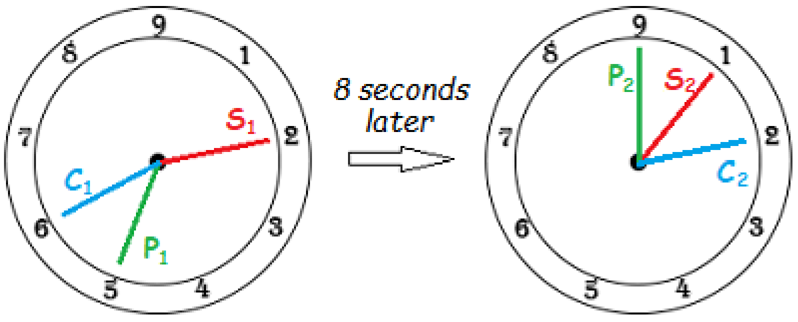
Nota: Se liberará una versión de los problemas en idioma español después de finalizar la competencia.

Problem A. A Strange Watch

Source file name:	Astrange.c, Astrange.cpp, Astrange.java, Astrange.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`José Manuel Salazar Meza - UFPS (Estudiante)`

Sebastian loves watches and today he decided to visit the old watchmaker's store. When Sebastian arrived, he was delighted with the large variety of styles and designs that he found there. However, today is not just any day, as the old watchmaker (who is a bit crazy) showed him a new watch with a design that he had been working on for years. It is a strange watch that works in the following way:

- The strange watch has the numbers from 1 to n written in a circular shape and three hands that we'll call S, P and C.
- The S hand moves to the next number in clockwise order each time one second elapses.
- The P hand moves to the next number in clockwise order if the S hand points to a **prime** number (both move at the same time).
- The C hand moves to the previous number in counter-clockwise order if the S hand points to a **composite** number (both move at the same time).



The image shows a strange watch with $n = 9$. The leftmost watch shows the S_1, P_1 and C_1 hands pointing at the numbers 2, 5 and 6 respectively. Then, 8 seconds later, the rightmost watch shows the S_2, P_2, and C_2 hands pointing at the numbers 1, 9, and 2 respectively.

Sebastian was so impressed by the design of the strange watch that he offered the old watchmaker all his savings to buy the watch. However, the old watchmaker did not agree to receive the money and instead told Sebastian that he would give him the watch for free if he guessed in which position each of the hands were when he arrived to the store. Can you help Sebastian win the strange watch?

Input

The first line contains two integers n and t ($2 \le n \le 10^6$, $0 \le t \le 10^{12}$) that indicate the amount of numbers written on the watch and the amount of seconds that have passed since Sebastian arrived at the store.

The second line contains three integers S_2, P_2 and C_2 ($1 \le S_2, P_2, C_2 \le n$) that indicate the current position of the S, P, and C hands respectively.

Output

Print a single line with three integers S_1, P_1 and C_1 ($1 \le S_1, P_1, C_1 \le n$) indicating the positions of the S, P and C hands respectively when Sebastian arrived to the old watchmarker's store.

Examples

Input	Output

`9 8`
`1 9 2`
	`2 5 6`

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

`10 1000`
`1 1 1`
	`1 1 1`

Note

A number n is **prime** if it is greater than 1 and has only two different positive divisors: 1 and n.

A number n is **composite** if it is greater than 1 and is **not** prime. That is, it has one or more positive divisors other than 1 and n.

Problem B. Buying Balloons

Source file name:	Buying.c, Buying.cpp, Buying.java, Buying.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`Carlos Fernando Calderón Rivero - UFPS (Estudiante)`

Professor Milton, coordinator of the SILUX research group, needs to buy some balloons to reward every team that solves a problem in the 2020 UFPS Programming Contest.

The Professor has some balloons left over from the trainings but doesn't know if there will be enough. Now he needs to determine how many balloons he should buy.

Milton is not good at making estimates, so he consults with Professor Hugo who is an expert at them. Hugo tells him that it's very likely that all the teams will be able to solve all the problems of the contest. Because of that, the difference will be in the time it takes each team to solve them. The most important thing is to have enough balloons.

Following on Hugo's prediction, help Milton determine the minimum number of balloons he must buy such that after giving them (equally) to all the teams he ends up with no balloons remaining.

Input

The first line contains an integer T indicating the number of test cases.

The next T lines contain two integers a and b ($0 < a, b < 10^9$) that correspond to the number of balloons Milton has and the number of teams registered, respectively.

Output

Print one line for each test case with a single number corresponding to the minimum number of balloons that Milton must buy.

Input	Output

4
5 2
20 6
1 4
15 5
	1
	4
	3
	0

Problem C. Catastrophe in XianLe

Source file name:	Catastrophe.c, Catastrophe.cpp, Catastrophe.java, Catastrophe.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`Crisel Jazmin Ayala Llanes - UFPS (Estudiante)`

The kingdom of XianLe is a prosperous and beautiful nation known for its great treasures in gold and jewels as well as in its culture, plenty of art and literature. For this reason, the terrible ghost Bai Wuxiang, *The faceless White*, motivated by his hatred and envy of this kingdom, has decided to throw a curse on the kingdom of XianLe: a catastrophe that will be unleashed in X days destroying everything in its way.

Fortunately, the heavens did not abandon this kingdom, and due to a miracle, some talismans have appeared in the sanctuaries of every city, which allow to protect against any kind of curse.

The kingdom of XianLe consists of N cities and M two-way roads distributed, so that it is always possible to go from a city to any other using these roads. If a person goes from city i to city j $(1 \le i, j \le N)$ using a road that connects those two cities, this journey will cost them a number of $t_{i,j}$ days. Each city i $(1 \le i \le N)$ has a population of P_i habitants and a number of Q_i talismans. A talisman can protect just one person.

With the hope of getting his people saved as soon as possible, the crown prince of XianLe has sought the help of a powerful programmer god. The prince wants to know what is the highest amount of inhabitants that can survive the catastrophe if he redistributes the population of each city in an optimal way and, besides that, what is the minimum amount of days that must pass so that those inhabitants are safe. Can you help him?

Input

The first line of input contains three integers N $(1 \le N \le 100)$, M $(N-1 \le M \le 5000)$ y X $(0 \le X \le 10^6)$ — the number of cities in the kingdom of XianLe, the number of roads and the number of days before the disaster takes place, respectively.

The next line contains N integers P_i $(0 \le P_i \le 100)$ separated by a single space, representing the population for each city from 1 to N.

The next line contains N integers Q_i $(0 \le Q_i \le 100)$ separated by a single space, representing the number of talismans for each city from 1 to N.

Each one of the following M lines contains three integers i, j $(1 \le i, j \le N)$ and $t_{i,j}$ $(1 \le t_{i,j} \le 10^5)$ indicating that there is a road between the cities i and j which costs $t_{i,j}$ days to go from a city to the other.

Output

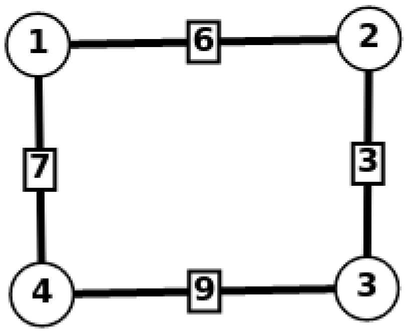
Print a single line with two integers, separated with a single space, indicating the highest number of survivors and the lowest number of days, respectively.

Input	Output

```
4 4 10
0 2 5 3
5 3 0 2
1 2 6
2 3 3
3 4 9
4 1 7
```
	`9 10`

Note

For the test case, a way to redistribute the population optimally is described below:



Bring 3 habitants of the city 4 to city 1,

Bring 2 habitants of the city 2 to city 1,

Bring 3 habitants of the city 3 to city 2 and

Bring 2 habitants of the city 3 to city 4,

Problem D. Digital Roots

Source file name:	Digital.c, Digital.cpp, Digital.java, Digital.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`José Manuel Salazar Meza - UFPS (Estudiante)`

Elizabeth loves competitive programming! Because of that, she has been training hard for an important competition. She wants to improve her math skills so she decided to solve many problems related to this topic. Among so many problems, she found one in which she had to calculate the digital root of a very large non-negative integer.

The digital root of a non-negative integer is defined as the value obtained by adding its digits repeatedly until the result of the addition is a single digit number. For example, the digital root of 34567 is 7 because $3 + 4 + 5 + 6 + 7 = 25$ and then $2 + 5 = 7$.

Elizabeth was able to solve that problem (very good Eli!). However, while analyzing some test cases on her board, she came up with an even more challenging problem.

Firts, she writes a very large non-negative integer on her board. Then, she applies a sequence of operations zero or more times. In one operation, she can erase a digit either at the beginning or end of the number.

Now she wants to know, for each possible digital root r $(0 \le r \le 9)$ what is the number of ways in which she can apply these operations and obtain a number of at least one digit whose digital root is equal to r. Elizabeth is close to solving this problem, Could you solve it too?

Input

The first line contains an integer d $(1 \le d \le 10^5)$ — the number of digits in the number.

The second line contains a non-negative integer n of d digits — the number written on the board.

Output

Print a single line containing the answer for each possible digital root r in ascending order by r. These values must be separated by a single space.

Examples

Input	Output

`5`
`34567`
	`0 0 1 2 3 1 2 3 0 3`

`9`
`311248370`
	`1 6 6 4 6 2 5 7 5 3`

`10`
`0987654321`
	`1 3 3 10 3 3 10 3 9 10`

Problem E. Efficient Healing

Source file name:	Efficient.c, Efficient.cpp, Efficient.java, Efficient.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`Carlos Fernando Calderón Rivero - UFPS (Estudiante)`

“Braveland Heroes” es un juego de estrategia basado en turnos, en el cual tienes que librar batallas contra otros jugadores usando diferentes tipos de personajes.

La sanadora es uno de los personajes más importantes del juego ya que una vez por batalla se puede activar su habilidad de sanación. Por facilidad de este problema vamos a usar únicamente este personaje y definiremos las siguientes características:

- $TropaInicial$: Cantidad de sanadoras con las que iniciamos la batalla.
- $TropaActual$: Cantidad de sanadoras que tenemos actualmente en la batalla.
- $PuntosDeVida$: Cantidad de vida que tiene cada sanadora.
- $PoderSanador$: Cantidad de vida que puede sanar cada sanadora de la tropa actual al activar la habilidad de sanación.

La habilidad de sanación consiste en que las sanadoras recuperan puntos de vida a las unidades que han sido derrotadas en batalla. La cantidad de $VidaRecuperada$ es igual a $TropaActual * PoderSanador$. Sin embargo, este valor está limitado por la cantidad de $VidaPerdida$, la cuál es igual a $(TropaInicial - TropaActual) * PuntosDeVida$. Por esto, vamos a definir el índice de recuperación IR como la cantidad real de vida recuperada por las sanadoras al activar su habilidad de sanación y cuyo valor es igual a $\min(VidaRecuperada, VidaPerdida)$.

Dados los valores de $TropaInicial$, $PuntosDeVida$ y $PoderSanador$, tu tarea es encontrar la cantidad de sanadores que debes tener en la batalla ($TropaActual$) tal que, al activar la habilidad especial, IR sea lo máximo posible.

Input

La entrada comienza con un entero T indicando el número de casos de prueba.

Luego siguen T líneas, cada una describiendo un caso de prueba con tres números enteros N, L $(1 \leq N, L \leq 10^9)$ y H $(1 \leq H \leq L)$ que corresponden a los valores de $TropaInicial$, $PuntosDeVida$ y $PoderSanador$, respectivamente.

Output

Por cada caso de prueba imprime una sola línea con un número entero indicando la respuesta al problema.

Examples

Input	Output

2
20 4 1
20 4 3
	16
	11

Problem F. Fun for Money

Source file name:	Fun.c, Fun.cpp, Fun.java, Fun.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`Gerson Yesid Lázaro Carrillo - UFPS (Graduado)`

Alice and Bob have reached the finale of the extreme games show "Fun for Money". The final challenge is overwhelmingly simple and scary:

The designers of the show have created a surprisingly big structure that is suspended with cranes at 200 meters high. The structure is composed by n platforms numbered from 1 to n joined together using $n-1$ ropes. Alice and Bob will have to test their balance to move between platforms walking on the ropes. The ropes could be used in any direction and have been disposed of in such a way that a path of ropes between any two platforms always exists.

At the beginning of the game, Alice and Bob must choose a different platform to start. Then, they will start walking on the ropes following the path that they want. Every time one of them leaves a platform (including the initial one), the platform closes and no one can use it again. Each walk one of them do on a rope from one platform to another will add one nlogonian dollar to their individual accumulation. They will try to keep playing while they have open platforms to move on (and obviously as long as they do not fall balancing on the ropes).

The game ends when they both leave the game, either due to falls or because they have no other platform to move to. Their final prize will be the multiplication of the amount of nlogonian dollars that they both accumulated. Your task is to calculate the maximum final prize that they can win if they play optimally.

Security notes from the show: It is guaranteed that Alice and Bob have a safety harness to protect them of any accident. Before starting the recording of the show, Alice, Bob, the presenter and the technical team have been tested for coronavirus and all of them got a negative result. During the recording, all of the biosecurity protocols are being followed. Because of that, Alice and Bob cannot be together on the same platform at any time to ensure physical distancing.

Input

The first line contains an integer n ($2 \le n \le 65536$) — the amount of platforms.

Then $n-1$ lines follow, each one with two integers a and b, representing a rope that connects the platforms a and b ($1 \le a, b \le n$).

Output

The output should contain a single line in the format "Alice and Bob won x nlogonian dollars", replacing x with the answer to the problem.

Examples

Input	Output

5
2 5
1 2
1 3
1 4
	`Alice and Bob won 2 nlogonian dollars`

3
1 2
2 3
	`Alice and Bob won 0 nlogonian dollars`

Problem G. GyM's Problem

Source file name:	Gyms.c, Gyms.cpp, Gyms.java, Gyms.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`José Manuel Salazar Meza - UFPS (Estudiante)`

Gerson and Melissa were two very recognized competitive programmers from UFPS. They are participating in this contest right now and are trying to solve this problem:

Given an array a of n positive integers, calculate the number of different pairs (a_i, a_j) with $i < j$.

Can you solve this problem before they do and show them who is the boss now?

Input

The first line contains a positive integer n $(1 \le n \le 10^5)$ — the size of the array.

The second line contains n positive integers a_i $(1 \le a_i \le 10^9)$ — the numbers in the array.

Output

Print a single line with a non-negative integer — the answer to the problem.

Examples

Input	Output
3 1 2 3	3
5 4 5 4 5 6	6
6 7 8 7 7 8 8	4

Note

In the first test case there are 3 pairs: $(1, 2)$, $(2, 3)$ and $(1, 3)$.

In the second test case there are 6 pairs: $(4, 5)$, $(4, 6)$, $(5, 6)$, $(4, 4)$, $(5, 4)$ and $(5, 5)$.

In the third test case there are 4 pairs: $(7, 8)$, $(7, 7)$, $(8, 8)$ and $(8, 7)$.

Problem H. Hidden Shapes

Source file name:	Hidden.c, Hidden.cpp, Hidden.java, Hidden.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second` / `256 megabytes`
Author(s):	`Carlos Fernando Calderón Rivero - UFPS (Estudiante)`

Luisa loves TV contests. Her favorite contest is called "Hidden Shapes". In this contest there are different types of challenges that consist of finding the number of hidden shapes that are among a lot of points. The only difference between the challenges is in the type of figure to look for, these can be squares, rectangles, triangles, etc.

The most difficult type of challenge in the contest is to find triangles, but it is also the one that gives the most money. Luisa has called twice trying to win the jackpot in this challenge but she has not been able to hit the correct answer. Because of that, she wants to be ready this time.

Luisa asks you to develop a program that, given a set of points, can find the number of different *valid triangles* that can be formed. A triangle is valid if its vertices are between the given points and its area is greater than 0. A triangle is different from another if at least one of its vertices is different from any of the other.

Input

The input begins with an integer T indicating the number of test cases. For each test case, the first line consists of an integer n $(3 \leq n \leq 100)$ indicating the number of points displayed on the screen.

The next n lines will contain two integers x and y $(-100 \leq x, y \leq 100)$ indicating the (x, y) coordinates of the points shown on the screen on a Cartesian plane. Is guaranteed that no two points are in the same coordinates.

Output

For each case, print a line with a non-negative integer indicating the number of triangles that Luisa must answer to win the jackpot.

Input	Output

2
4
0 0
0 1
1 0
1 1
4
0 0
3 3
-1 -1
1 2
	4
	3

Problem I. It's a Palindromic Trip

Source file name:	Its.c, Its.cpp, Its.java, Its.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`2 seconds / 256 megabytes`
Author(s):	`José Manuel Salazar Meza - UFPS (Estudiante)`

When Crisel and Carlos went to Buenos Aires, Argentina, they took advantage of their stay there to visit wonderful places such as The Obelisco, Caminito, The Memory Park, among many others.

In order to avoid getting lost, they drew a map with the n places they wanted to visit and labeled each place with the initial letter of its respective name. Also, they drew $n-1$ two-way roads connecting pairs of places in such a way that it was possible to move between any two places using only these roads (possibly, passing through some intermediate places).

Nowadays, they are bored at home due to the quarantine. However, today Crisel found the map they had made and came up with a rounds game, so she called Carlos to tell him her idea.

There are two types of rounds, in a type 1 round, each player chooses one of the n places of the map. Then, they form a string consisting of the labels on each of the places they must pass in order to go from Crisel's chosen place to Carlos chosen place (including both players chosen places). If the resulting string is a **palindrome**, then it's said to be a *palindromic trip*! The first to say whether or not it's a palindromic trip wins the round.

In a type 2 round, Crisel can change the label of a place in the map with another randomly chosen letter.

Now that you also have the map, could you tell, for each type 1 round, if it's a palindromic trip?

Input

The first line contains two integers n and r ($2 \le n \le 10^5$, $1 \le r \le 10^5$) — the number of places on the map and the number of rounds, respectively.

The next line contains n lowercase English letters, which correspond to the initial labels of each place (from place 1 to place n).

Each of the following $n-1$ lines contains two integers u and v ($1 \le u, v \le n$, $u \ne v$), indicating that there is a road connecting places u and v.

The following r lines contain the description of each round:

- A type 1 round begins with a 1 followed by two integers u and v ($1 \le u, v \le n$) indicating that Crisel chose the place u and Carlos the place v.
- A type 2 round begins with a 2 followed by an integer x and a character c ($1 \le x \le n$, c is a lowercase English letter) indicating that Crisel set the label of the place x to be the letter c.

Rounds are given in the order in which they occur. It is guaranteed that the last round is of type 1.

Output

For each type 1 round, print "Yes" if it's a palindromic trip, otherwise print "No" (without the quotes).

Examples

Input	Output
4 5 ufps 1 2 3 1 4 1 1 2 4 2 3 s 1 3 4 2 1 f 1 2 1	No Yes Yes
7 6 abacaba 1 4 2 1 2 5 1 6 7 6 3 6 1 5 3 1 4 6 2 6 c 1 4 6 1 7 5 1 3 7	Yes No Yes No Yes

Note

Use fast I/O methods

In the first case, the strings written in each type 1 round are respectively:

- "`fus`" or "`suf`".
- "`sus`".
- "`ff`".

A string is called a **palindrome** if it reads the same from left to right and from right to left. For example "`abacaba`", "`cc`" and "`m`" are palindromes, but "`go`", "`reloaded`" and "`equipo`" are not.

Problem J. Just an Emergency

Source file name:	Just.c, Just.cpp, Just.java, Just.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`4 second / 256 megabytes`
Author(s):	`Hugo Humberto Morales Peña - UTP (Profesor)`

En días pasados el profesor *Humbertov Moralov* sufrió un pequeño accidente en su pie izquierdo y tuvo que ir al hospital para que lo "atendieran" por urgencias. Durante las casi cuatro horas de espera, el profesor tuvo tiempo para analizar y entender cómo funciona el servicio de urgencias colombiano.

Constantemente están llegando al servicio de urgencias pacientes que solicitan la atención médica, los cuales inicialmente son valorados por un médico según un método denominado Triage en el cual se determina la prioridad de la urgencia. Las posibles clasificaciones del Triage son 1, 2, 3, 4 y 5, siendo 1 la calificación con la cual se indica que la atención tiene que ser inmediata (la vida del paciente está en alto riesgo) y 5 la calificación con la cual se indica que el paciente necesita atención médica pero que no tiene comprometido ningún órgano vital y que por lo tanto puede esperar por cinco o seis horas en la sala de urgencias. Después de la valoración del médico los pacientes pasan a la sala de espera y allí son llamados por orden de prioridad con respecto a la clasificación del Triage.

El profesor Humbertov Moralov requiere de su ayuda y le pide a usted como estudiante de un programa académico de *Ciencias de la Computación* una solución computacional para poder determinar en qué orden son atendidos los pacientes en el servicio de urgencias.

Nota: Cuando hay varios pacientes con la misma prioridad en el Triage se tiene que atender al que llego primero (con respecto a la hora de llegada) ... si esto no se respeta se genera una "asonada" por parte de los pacientes y el servicio de urgencias es destruido! ... de usted depende que el servicio de urgencias siga prestando su servicios a la comunidad!

Input

La entrada del problema consiste de varios casos de prueba. Cada caso de prueba comienza con una línea que contiene un número entero positivo n ($3 \leq n \leq 2 \cdot 10^5$) que representa el total de "transacciones" realizadas en el servicio de urgencias. Luego se presentan n líneas cada una de ellas comenzando con un par de números enteros positivos t h ($t \in \{1, 2\}$, $1 \leq h \leq 8 \cdot 10^5$) que representan respectivamente el tipo de transacción (1 para indicar que es una solicitud del servicio de urgencias de un paciente y 2 para indicar la atención de la urgencia por parte de un doctor) y la hora (en segundos). Para las "transacciones" del tipo 1 la línea se complementa con p s ($p \in \{1, 2, 3, 4, 5\}$, s es una cadena sin espacios que solo incluye letras mayúsculas del alfabeto inglés con una longitud máxima de 20) que representan respectivamente la prioridad del triage y el nombre del paciente. El final de los casos de prueba es dado por el fin de archivo (End Of File - EOF).

Output

Para cada caso de prueba, su programa debe imprimir tantas líneas como "transacciones" del tipo 2 tenga, cada una de estas líneas debe contener tres números enteros positivos y una cadena a b c s ($1 \leq a, b, c \leq 8 \cdot 10^5$, $a < b$) que representan la información de la atención al paciente con respecto a la hora de llegada, la hora en que es atendido, el tiempo total de espera y el nombre.

Examples

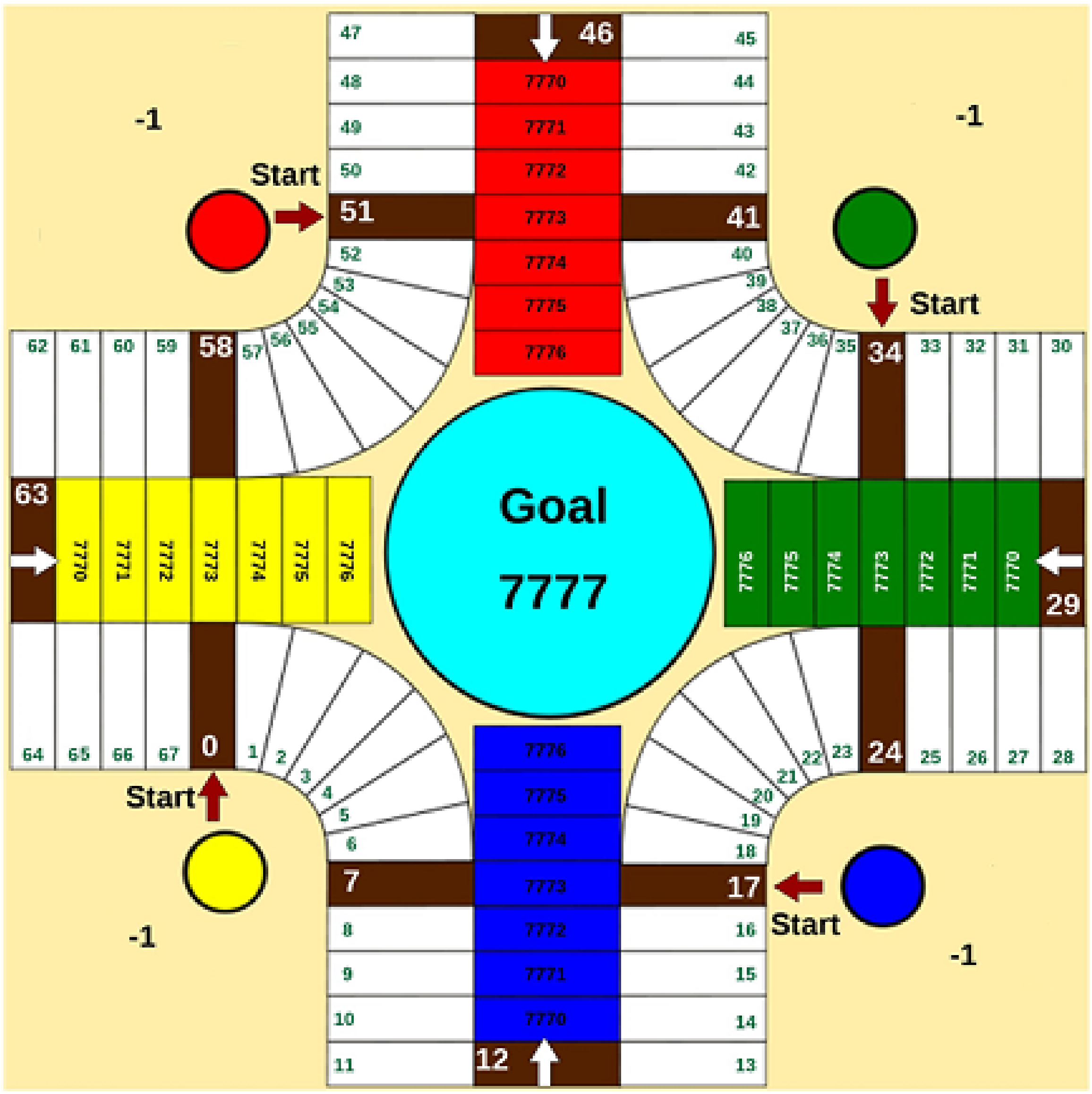
Input	Output
10 1 1 5 CARLOS 1 3 4 MANUEL 1 4 4 ANDRES 1 6 1 GABRIEL 2 9 1 13 2 YEFRI 2 20 1 24 1 STEVEN 2 30 2 50	6 9 3 GABRIEL 13 20 7 YEFRI 24 30 6 STEVEN 3 50 47 MANUEL

Use fast I/O methods

Problem K. K-Parchis World Cup

Source file name:	Kparchis.c, Kparchis.cpp, Kparchis.java, Kparchis.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`2 second / 256 megabytes`
Author(s):	`Milton Jesús Vera Contreras - UFPS (Profesor)`

Parchis is a well-known game that everyone knows how to play. It consists of a board, two dice, and one or more pieces for each player. In this case only one piece will be used. A Parchis board can be for 4 players, for 5 players, for 6 players, etc. The 4-player board is the baseline and its design is replicated for any number of players. The figure shows the layout for a 4-player board and the proposed coding to represent the board on a computer.



The board has four zones::

1. The first zone is the start of the game and each player has his private zone. It is coded with −1.

To get out of this zone, the player must roll a double number with the dice (11,22,33,44,55 or 66). And it will go to the box marked with an arrow as the start.

2. The second zone is the goal of the game, the first player to arrive wins. It is encoded with 7777.
3. The third zone is general and has 17 places for each player. A board for 4 players has 68 places, for 5 players it has 85 places, for 10 players it has 170 places and so on. In the figure, these places are white and brown. It is coded from 0 to $17 * K\breve{}1$ (K players). The white places only allow one piece. If there is a collision of two pieces, the arriving piece makes the other return to the first area and start the game again. The brown places are safe and allow multiple pieces.
4. The fourth zone is safe and each player has their private zone with a custom color. It is encoded with numbers from 7770 to 7776: the way to the goal.

The rules of the game are very simple:

1. The game is turn-based. A player on his turn must roll the dice and move his token according to the following rules. The player next to position zero (0) always starts. Then the turns follow the order in which the third zone is listed.
2. To exit the first zone, the player must roll a double number with the dice (11,22,33,44,55 or 66).
3. If a player rolls the dice and rolls a double number, he rolls again.
4. If the player is in the third zone, he advances as many places as the sum of the values obtained when rolling the dice.
5. If the player is in the fourth zone and the sum of the values obtained by rolling the dice is greater than what is necessary to reach the goal then he must advance and return. For example, if the player is at 7770 and when rolling the dice they get a 65 then they advance 11 places, thus reaching the goal 7777 and returning to 7773.
6. If the player is in the fourth zone and the sum of the values obtained by rolling the dice is equal to what is necessary to reach the goal then the player wins and the game ends. The player who wins is placed in position 7777.
7. If there is no winner, each player rolls the dice in order again and follows the rules of the game.

You want to develop a computer program that receives a list of dice rolls for a game with any number of players K. The program must report if there was a winner, which was the winner and the sequence of places on the board by each player after each roll of the dice.

Input

The input consists of multiple test cases. Each test case consists of three lines:

The first line contains an integer K ($4 \leq K < 100$) indicating the number of players.

The second line contains an integer N ($K \leq N < 10^5$) that indicates the number of dice rolls.

And the third line contains N two-digit numbers separated by a single space, indicating each of the dice rolls. Each of the digits corresponds to the result of a die. Only the digits 1, 2, 3, 4, 5 and 6 are guaranteed to be used. The end of the test cases is the end of file (EOF).

Output

For each test case $K + 1$ lines must be printed.

The first K lines correspond to the numbers of the places occupied by each player during the game, including the starting area of the game -1 and the repetitions of squares that may be presented, using the following format:

Each line begins with the word "`Player`" followed by a space and the player's number (from 1 to K). Then another space, the symbol "`=`", another space and finally a sequence of numbers enclosed in braces, separated by commas and without spaces between them:

"`Player # = {-1,casilla-1,casilla-2,...,casilla-n-2,casilla-n-1,casilla-n}`".

The last line (line $K + 1$) reports the winner "`The winner is Player W`" (where W is the number of the winning player) or reports that there is no winner yet "`There is no winner yet`". If there is a winner, that player is guaranteed to win the game by rolling N dice.

Examples

Input	Output

4
13
66 66 65 55 14 66 32 44 61 66 66 66 66
	Player 1 = -1,0,12,23,35,47,59,7777
	Player 2 = -1,17,22
	Player 3 = -1,34,39
	Player 4 = -1,51,58
	The winner is Player 1

4
12
66 23 55 14 66 32 44 61 66 65 51 23
	Player 1 = -1,0,5,17,28,-1
	Player 2 = -1,17,22,28
	Player 3 = -1,34,39,44
	Player 4 = -1,51,58
	There is no winner yet

4
13
66 55 65 55 14 66 32 41 66 66 66 66 66
	Player 1 = -1,0,10,21,33,45,57,7775,61
	Player 2 = -1,17,22
	Player 3 = -1,34,39
	Player 4 = -1,-1
	There is no winner yet

Note

Use fast I/O methods

Problem L. Lazy Team

Source file name:	Lazyteam.c, Lazyteam.cpp, Lazyteam.java, Lazyteam.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`Carlos Fernando Calderón Rivero - UFPS (Estudiante)`

The Lazy Team is a team that hates long-statement problems. Without many aspirations and with much laziness they decide to only read and try the short-statement problems (statements with less than one page). Although they don't see themselves as the future regional champions, they also don't want to be left blank in the contest.

A contest consist of problems of three different difficulties (easy, medium, and hard). Let's denote the number of easy, medium, and hard problems as a, b, and c respectively. Thus there's a total of $a + b + c$ problems. Also, let's denote the number of short-statement problems as n $(0 \leq n \leq a + b + c)$.

Before the contest, the team decided to use their computer to do some simulations. They want to know what is the probability of receiving at least k easy problems among the n short-statement problems of the contest. They can assume that the probability of a problem having a short statement is the same for all the problems.

Since they are so lazy, they are also lazy to do this calculation, can you help them?

Input

The input begins with an integer T indicating the number of simulations.

Each of the following T lines include a simulation. A simulation consists of five non-negative integers a, b, c $(1 \leq a + b + c \leq 16)$, n $(0 \leq n \leq a + b + c)$ and k $(0 \leq k \leq \min(a, n))$ that correspond to the number of easy, medium and hard problems, the number of short-statement problems and the number of easy problems they expect to choose, respectively.

Output

For each simulation, print a line with the answer to the problem rounded to four digits after the decimal point.

Input	Output

`4`
`8 3 1 2 2`
`3 5 2 3 1`
`9 0 1 5 3`
`1 1 4 1 1`
	`0.4242`
	`0.7083`
	`1.0000`
	`0.1667`

Problem M. Minesweeper App

Source file name:	Minesweeper.c, Minesweeper.cpp, Minesweeper.java, Minesweeper.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes`
Author(s):	`José Manuel Salazar Meza - UFPS (Estudiante)`

This problem is the same as "Once Again: Minesweeper App". The only difference between the two versions is in the constraints on the size of the board.

Natalia and Fernando work in a prestigious company called WBOSS. A few days ago, their boss assigned them a new project: Developing a mobile application of the famous game "Minesweeper".

Each level of the game will be a rectangular board with n rows and m columns. Each of the different $n \times m$ cells of the board can contain either a number or a mine. The cells that will contain numbers are already defined by the client but the cells that will contain mines are not.

The game must be easy, so the board of each level must have the minimum possible number of mines and it must be a *valid board.* A board T is valid if for each cell at position (i, j) that contains a number $(1 \leq i \leq n; 1 \leq j \leq m)$, there are exactly $T_{i,j}$ mines in its adjacent cells (in all eight possible directions). We denote $T_{i,j}$ as the number that is in the cell at position (i, j) of the board T.

Finally, the game must have the largest amount of levels and all of them must be different. Two levels are different if there is at least one cell that contains a mine on one of the boards but not on the other.

Natalia quit from WBOSS for a very high paying offer, so the project is now in the hands of Fernando. Could you help him calculate the minimum number of mines that a board must have in order to be considered valid? In addition to that, Could you determine how many different levels can he make with this number of mines?

Input

The first line contains two integers n and m $(1 \leq n * m \leq 20)$ that indicate the number of rows and columns on each board T in the game, respectively.

Then n lines follow each with m characters. Each character is either a "`.`" or a digit d $(1 \leq d \leq 8)$. If the j-th character of the i-th line is "`.`", then the cell at position (i, j) is empty. Otherwise, the digit d represents the value for $T_{i,j}$.

Output

The output consists of a single line. If it is impossible to develop the game print "`0`" (without the quotes).

Otherwise, prints two integers a and b — the number of levels and the number of mines, respectively.

Examples

Input	Output

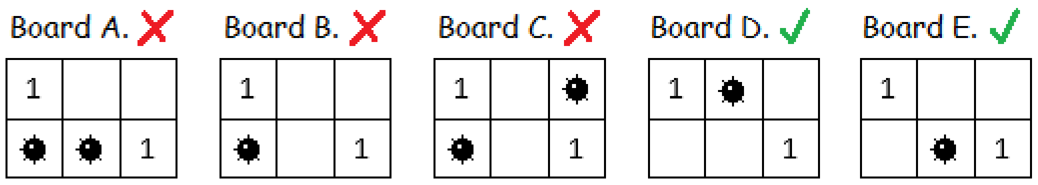
`2 3`
`1..`
`..1`
	`2 1`

`3 4`
`1...`
`2...`
`...3`
	`4 5`

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

Note

For the first test case, the following graphic shows five sample boards:



Board A is not valid because the cell $A_{1,1} = 1$, but there are 2 mines in its adjacent cells.

Board B is not valid because the cell $B_{2,3} = 1$, but there are no mines in its adjacent cells.

Board C is a valid board but does not have the minimum possible number of mines.

Boards D and E are valid and have the minimum possible number of mines. Also, they are different.

Problem N. Newbie to Master Journey

Source file name:	Newbie.c, Newbie.cpp, Newbie.java, Newbie.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`5 seconds / 1024 megabytes`
Author(s):	`Juan Camilo Bages - UniAndes (Graduado)`

Manuel is currently training to master the ancient and sacred art of "Programming Competitions". In order to do this, he has a master called JCB that he admires a lot and that gives him a training plan for preparing himself. JCB training is similar to Mister Miyagi's training to Daniel San which included a set of tasks such as waxing the car or paining the fence. Now JCB has assigned a mission to Manuel that not even Mister Miyagi would have dare to designate. Can Manuel complete this mission on time and with that dominate the next level of the "Programming Competitions"?

The mission starts with an array of N strings and a sacred parchment that Manuel has to find in the deepest parts of the Himalayas. Once obtained, Manuel must complete a series of Q tasks written in the sacred parchment. Each of these tasks consist of an interval $[L, R]$. However, the parchment can only be read by those who are worth it. Because of this, the parchment has two values A and B for each task and Manuel must use them to calculate the actual interval $[L, R]$ in the following way:

- $L = (Last + A - 1) \bmod N + 1$
- $R = (Last + B - 1) \bmod N + 1$

In the case that $L > R$, Manuel can simply swap the values of L and R such that the condition $L \leq R$ always holds. *Last* represents the answer of the previous task and can be assumed to be 0 before the first task.

Manuel asks his master JCB what is the task that must be completed with this interval. To this JCB answers the following in a very abstract and confusing way:

"The answer of each task can be found in the bottom of your heart. Look inside you and then calculate the number of distinct substrings that are in the interval of the array from index L up to R (including both L and R)."

Manuel is still confused because the master talks in a very strange way so he goes to the Oracle for more information. Manuel gives the Oracle some gold coins and the Oracle very kindly shows Manuel a series of examples about the tasks that he must complete and an explanation of these.

Input

The input starts with two values N ($1 \leq N \leq 5000$) and Q ($1 \leq Q \leq 10^6$) representing the size of the array A and the number of tasks that must be completed respectively.

Then you'll recieve N strings A_i that represent the elements of the array. You can assume that the sum of the length of all strings A_i is ≤ 5000 (i.e. $\sum_{i=1}^{N} |A_i| \leq 5000$). Each string consists only of lowercase English letters ($a - z$).

After this, you'll recieve Q lines, each one including two numbers A ($1 \leq A \leq N$) and B ($1 \leq B \leq N$) that represent each of the tasks that Manuel must complete.

Output

Print Q lines each one with a number V_i representing the answer for the ith task.

Input	Output
3 6	15
banana	15
ana	16
j	1
1 1	16
1 2	15
1 3	
2 2	
2 3	
3 3	

Note

The substrings of each item in the array are the following:

`banana` $\rightarrow$ {`a`, `an`, `ana`, `anan`, `anana`, `b`, `ba`, `ban`, `bana`, `banan`, `banana`, `n`, `na`, `nan`, `nana`}

`ana` $\rightarrow$ {`a`, `an`, `ana`, `n`, `na`}

`j` $\rightarrow$ {`j`}

Each task is solved the following way:

$A = 1, B = 1, Last = 0 \rightarrow L = 1, R = 1$

subarray $\rightarrow$ {`banana`}

15

$A = 1, B = 2, Last = 15 \rightarrow L = 1, R = 2$

subarray $\rightarrow$ {`banana`, `ana`} (all substrings of ana are substrings of banana)

15

$A = 1, B = 3, Last = 15 \rightarrow L = 1, R = 3$

subarray $\rightarrow$ {`banana`, `ana`, `j`} (all substrings of ana are substrings of banana)

16

$A = 1, B = 1, Last = 16 \rightarrow L = 2, R = 2$

subarray $\rightarrow$ {`ana`}

5

$A = 3, B = 1, Last = 5 \rightarrow L = 2, R = 3$

subarray $\rightarrow$ {`ana`, `j`}

6

$A = 3, B = 3, Last = 6 \rightarrow L = 3, R = 3$

subarray $\rightarrow$ {`j`}

1

Problem O. Once Again: Minesweeper App

Source file name:	Onceagain.c, Onceagain.cpp, Onceagain.java, Onceagain.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1.5 seconds / 256 megabytes`
Author(s):	`José Manuel Salazar Meza - UFPS (Estudiante)`

This problem is the same as "Minesweeper App". The only difference between the two versions is in the constraints on the size of the board.

Natalia and Fernando work in a prestigious company called WBOSS. A few days ago, their boss assigned them a new project: Developing a mobile application of the famous game "Minesweeper".

Each level of the game will be a rectangular board with n rows and m columns. Each of the different $n \times m$ cells of the board can contain either a number or a mine. The cells that will contain numbers are already defined by the client but the cells that will contain mines are not.

The game must be easy, so the board of each level must have the minimum possible number of mines and it must be a *valid board*. A board T is valid if for each cell at position (i, j) that contains a number $(1 \leq i \leq n; 1 \leq j \leq m)$, there are exactly $T_{i,j}$ mines in its adjacent cells (in all eight possible directions). We denote $T_{i,j}$ as the number that is in the cell at position (i, j) of the board T.

Finally, the game must have the largest amount of levels and all of them must be different. Two levels are different if there is at least one cell that contains a mine on one of the boards but not on the other.

Natalia quit from WBOSS for a very high paying offer, so the project is now in the hands of Fernando. Could you help him calculate the minimum number of mines that a board must have in order to be considered valid? In addition to that, Could you determine how many different levels can he make with this number of mines?

Input

The first line contains two integers n and m $(1 \leq n * m \leq 60)$ that indicate the number of rows and columns on each board T in the game, respectively.

Then n lines follow each with m characters. Each character is either a "`.`" or a digit d $(1 \leq d \leq 8)$. If the j-th character of the i-th line is "`.`", then the cell at position (i, j) is empty. Otherwise, the digit d represents the value for $T_{i,j}$.

Output

The output consists of a single line. If it is impossible to develop the game print "`0`" (without the quotes).

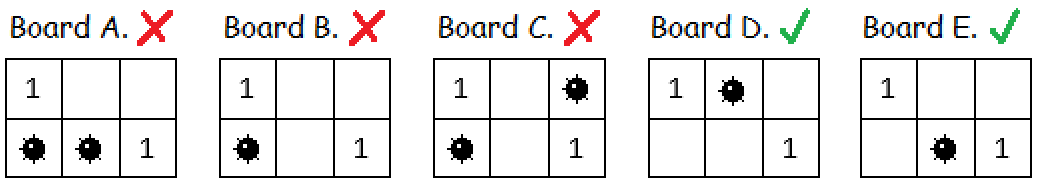
Otherwise, prints two integers a and b — the number of levels and the number of mines, respectively.

Examples

Input	Output
`2 3`	`2 1`
`1..`	
`..1`	
`3 4`	`4 5`
`1...`	
`2...`	
`...3`	
`3 2`	`0`
`1.`	
`.3`	
`3.`	

Note

For the first test case, the following graphic shows five sample boards:



Board A is not valid because the cell $A_{1,1} = 1$, but there are 2 mines in its adjacent cells.

Board B is not valid because the cell $B_{2,3} = 1$, but there are no mines in its adjacent cells.

Board C is a valid board but does not have the minimum possible number of mines.

Boards D and E are valid and have the minimum possible number of mines. Also, they are different.