Maratón de Programación 2020

12/12

Java

python

C/C++

RED DE PROGRAMACIÓN COMPETITIVA

RPC

```
while (true) {
    keepTraining();
}
```

SILUX

"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

YouTube

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.

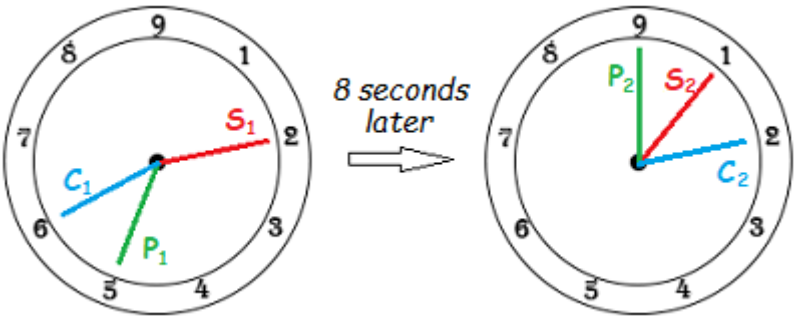
Nota: Se liberará una versión de los problemas en idioma español después de finalizar la competencia.

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)`

A Sebastian le encantan los relojes y hoy él decidió visitar la tienda del viejo relojero. Cuando Sebastian llegó, quedó encantado con la gran variedad de estilos y diseños que encontró allí. Sin embargo, hoy no es un día cualquiera, ya que el viejo relojero (que está un poco loco) le mostró un nuevo reloj con un diseño en el que había estado trabajando por años. Es un extraño reloj que funciona de la siguiente manera:

- El extraño reloj tiene los números del 1 al n escritos en forma circular y tres manecillas que llamaremos S, P y C.
- La manecilla S se mueve al número siguiente en sentido horario cada vez que transcurre un segundo.
- La manecilla P se mueve al número siguiente en sentido horario si la manecilla S apunta a un número **primo** (ambas se mueven a la vez).
- La manecilla C se mueve al número anterior en sentido antihorario si la manecilla S apunta a un número **compuesto** (ambas se mueven a la vez).



La imagen muestra un extraño reloj con $n = 9$. El reloj más a la izquierda muestra las manecillas S_1, P_1 y C_1 apuntando a los números 2, 5 y 6 respectivamente. Luego, 8 segundos después, el reloj más a la derecha muestra las manecillas S_2, P_2 y C_2 apuntando a los números 1, 9 y 2 respectivamente.

Sebastian quedó tan impresionado con el diseño del extraño reloj que le ofreció al viejo relojero todos sus ahorros para comprar el reloj. Sin embargo, el viejo relojero no aceptó recibir el dinero y en cambio le dijo a Sebastian que le daría el reloj gratis si adivinaba en qué posición estaba cada una de las manecillas cuando él llegó a la tienda. ¿Puedes ayudar a Sebastian a ganar el extraño reloj?

Input

La primera línea contiene dos enteros n y t ($2 \leq n \leq 10^6$, $0 \leq t \leq 10^{12}$) que indican la cantidad de números escritos en el reloj y la cantidad de segundos que han pasado desde que Sebastian llegó a la tienda.

La segunda línea contiene tres enteros S_2, P_2 y C_2 ($1 \leq S_2, P_2, C_2 \leq n$) que indican la posición actual de las manecillas S, P y C respectivamente.

Output

Imprime una sola línea con tres enteros S_1, P_1 y C_1 ($1 \leq S_1, P_1, C_1 \leq n$) indicando las posiciones de las manecillas S, P y C respectivamente cuando Sebastian llegó a la tienda del viejo relojero.

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

Un número n es **primo** si es mayor que 1 y tiene solo dos divisores positivos diferentes: 1 y n.

Un número n es **compuesto** si es mayor que 1 y **no** es primo. Es decir, tiene uno o más divisores positivos aparte de 1 y 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)`

El profesor Milton, coordinador del semillero de investigación SILUX, necesita comprar algunos globos para premiar a cada equipo que resuelva un problema en la Maratón de Programación UFPS 2020.

El profesor tiene algunos globos que sobraron de los entrenamientos pero no sabe si serán suficientes. Ahora necesita determinar cuántos globos debe comprar.

Milton no es bueno para hacer estimaciones, por lo que consulta con el profesor Hugo quien es un experto en ellas. Hugo le dice que es muy probable que todos los equipos puedan resolver todos los problemas de la competencia. Por eso, la diferencia estará en el tiempo que tarde cada equipo en resolverlos. Lo más importante es tener suficientes globos.

Siguiendo la predicción de Hugo, ayuda a Milton a determinar la mínima cantidad de globos que debe comprar de tal manera que después de dárselos (por igual) a todos los equipos, no quede ningún globo.

Input

La primera línea contiene un entero T que indica el número de casos de prueba.

Las siguientes T líneas contienen dos enteros a y b ($0 < a, b < 10^9$) que corresponden a la cantidad de globos que tiene Milton y la cantidad de equipos inscritos, respectivamente.

Output

Imprime una línea por cada caso de prueba con un solo número correspondiente a la mínima cantidad de globos que Milton debe comprar.

Examples

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 Llanez - UFPS (Estudiante)`

El reino de XianLe es una próspera y bella nación reconocida por sus grandes riquezas tanto en oro y joyas como en su cultura, llena de arte y literatura. Es por esto que el terrible fantasma Bai Wuxiang, *El blanco sin rostro*, motivado por su odio y envidia al reino, decidió lanzar una maldición al reino de XianLe: una catástrofe que se desatará en X días destruyendo todo a su paso.

Por fortuna los cielos no abandonaron a este reino, y debido a un milagro, aparecieron unos talismanes en los santuarios de cada ciudad que permiten protegerse contra cualquier tipo de maldición.

El reino de XianLe consta de N ciudades y M caminos bidireccionales distribuidos de tal manera que siempre es posible ir de una ciudad a cualquier otra usando estos caminos. Si una persona va desde la ciudad i a la ciudad j $(1 \leq i, j \leq N)$ utilizando un camino que conecta estas dos ciudades, este viaje le costará una cantidad de $t_{i,j}$ días. Cada ciudad i $(1 \leq i \leq N)$ tiene una población de P_i habitantes y una cantidad de Q_i talismanes. Un talismán puede proteger únicamente a una sola persona.

Con el deseo de salvar a su gente lo más pronto posible, el príncipe heredero de XianLe ha solicitado la ayuda de un poderoso dios programador. El príncipe quiere saber cuál es la máxima cantidad de habitantes que pueden sobrevivir a la catástrofe si redistribuye a la población de cada ciudad de manera óptima y, además de eso, cuál es la mínima cantidad de días que deben pasar para que esos habitantes estén a salvo. ¿Puedes ayudarlo?

Input

La primera línea de entrada contiene tres números enteros N $(1 \leq N \leq 100)$, M $(N - 1 \leq M \leq 5000)$ y X $(0 \leq X \leq 10^6)$ — el número de ciudades del reino de XianLe, el número de caminos y la cantidad de días antes de que ocurra la catástrofe, respectivamente.

La siguiente línea contiene N números enteros P_i $(0 \leq P_i \leq 100)$ separados por un solo espacio, que representan la población para cada ciudad de 1 a N.

La siguiente línea contiene N números enteros Q_i $(0 \leq Q_i \leq 100)$ separados por un solo espacio, que representan la cantidad de talismanes para cada ciudad de 1 a N.

Cada una de las siguientes M líneas contiene tres enteros i, j $(1 \leq i, j \leq N)$ y $t_{i,j}$ $(1 \leq t_{i,j} \leq 10^5)$ indicando que hay un camino entre las ciudades i y j que cuesta $t_{i,j}$ días para ir de una de las ciudades a la otra.

Output

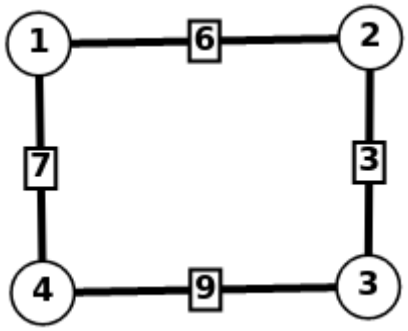
Imprime una sola línea con dos enteros, separados por un solo espacio, indicando la máxima cantidad de sobrevivientes y la mínima cantidad de días respectivamente.

Examples

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

Para el caso de prueba, a continuación se describe una forma de redistribuir a la población de manera optima:



Llevar 3 habitantes de la ciudad 4 a la ciudad 1,

Llevar 2 habitantes de la ciudad 2 a la ciudad 1,

Llevar 3 habitantes de la ciudad 3 a la ciudad 2 y

Llevar 2 habitantes de la ciudad 3 a la ciudad 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)`

¡A Elizabeth le encanta la programación competitiva! Por eso, ella ha estado entrenando fuertemente para una importante competencia. Ella quiere mejorar sus habilidades en matemáticas así que decidió resolver muchos problemas relacionados con este tópico. Entre tantos problemas ella se encontró con uno en el que debía calcular la raíz digital de un número entero no negativo muy grande.

La raíz digital de un número entero no negativo se define como el valor obtenido de sumar sus dígitos repetidamente hasta que el resultado de la suma sea un número de un solo dígito. Por ejemplo, la raíz digital de 34567 es 7 porque $3 + 4 + 5 + 6 + 7 = 25$ y luego, $2 + 5 = 7$.

Elizabeth pudo resolver ese problema (¡muy bien Eli!). Sin embargo, mientras analizaba algunos casos de prueba en su tablero, se le ocurrió un problema aún más desafiante.

Primero que todo, ella escribe un número entero no negativo muy grande en su tablero. Luego, aplica una secuencia de operaciones cero o más veces. En una operación, ella puede borrar un dígito al inicio o al final del número.

Ahora ella quiere saber, para cada posible raíz digital r $(0 \leq r \leq 9)$ cuál es la cantidad de formas en las que puede aplicar estas operaciones y obtener un número de al menos un dígito cuya raíz digital sea igual a r. Elizabeth está cerca de resolver este problema, ¿podrías resolverlo también?.

Input

La primera línea contiene un entero d $(1 \leq d \leq 10^5)$ — la cantidad de dígitos del número.

La segunda línea contiene un entero no negativo n de d dígitos — el número escrito en el tablero.

Output

Imprime una sola línea conteniendo la respuesta para cada posible raíz digital r en orden ascendente por r. Estos valores deben estar separados por un solo espacio.

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 y Bob han llegado a la final del programa de concursos extremos "Diversión por dinero". El desafío final es abrumadoramente simple y aterrador:

Los diseñadores del espectáculo han creado una estructura sorprendentemente grande que se suspende con grúas a 200 metros de altura. La estructura está compuesta por n plataformas numeradas desde 1 hasta n, unidas por $n-1$ cuerdas. Alice y Bob tendrán que poner a prueba su equilibrio para moverse entre plataformas caminando sobre las cuerdas. Las cuerdas se pueden utilizar en cualquier dirección y se han dispuesto de tal manera que siempre existe un camino de cuerdas entre cualquier par de plataformas.

Al comienzo del juego, Alice y Bob deben elegir una plataforma diferente para comenzar. Luego, comenzarán a caminar sobre las cuerdas siguiendo el camino que quieran. Cada vez que uno de ellos sale de una plataforma (incluida la inicial), la plataforma se cierra y nadie puede volver a usarla. Cada caminata que haga uno de ellos sobre una cuerda de una plataforma a otra sumará un dólar nlogoniano a su acumulado individual. Ellos intentarán seguir jugando mientras tengan plataformas abiertas para avanzar (y obviamente mientras no caigan balanceándose sobre las cuerdas).

El juego termina cuando ambos abandonan el juego, ya sea por caídas o porque no tienen otra plataforma a la que moverse. Su premio final será la multiplicación de la cantidad de dólares nlogonianos que ambos acumularon. Tu tarea es calcular el máximo premio final que ellos pueden ganar si juegan óptimamente.

Notas de seguridad del programa: Se garantiza que Alice y Bob tienen un arnés de seguridad para protegerlos de cualquier accidente. Antes de iniciar la grabación del programa, Alice, Bob, el presentador y el equipo técnico han sido examinados por coronavirus y todos han obtenido resultado negativo. Durante la grabación, se siguen todos los protocolos de bioseguridad. Por eso, Alice y Bob no pueden estar juntos en la misma plataforma en ningún momento para asegurar el distanciamiento físico.

Input

La primera línea contiene un número entero n ($2 \leq n \leq 65536$) — la cantidad de plataformas.

Luego siguen $n-1$ líneas, cada una con dos enteros a y b, que representan una cuerda que conecta las plataformas a y b ($1 \leq a, b \leq n$).

Output

La salida debe contener una sola línea en el formato "Alice and Bob won x nlogonian dollars", reemplazando x por la respuesta al problema.

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 y Melissa fueron dos programadores competitivos muy reconocidos de la UFPS. Ellos están participando en esta competencia justo ahora y están intentando resolver este problema:

Dado un arreglo a de n enteros positivos, calcula el número de pares diferentes (a_i, a_j) con $i < j$.

¿Puedes resolver este problema antes que ellos y mostrarles quién manda ahora?

Input

La primera línea contiene un entero positivo n $(1 \leq n \leq 10^5)$ — el tamaño del arreglo.

La segunda línea contiene n enteros positivos a_i $(1 \leq a_i \leq 10^9)$ — los números del arreglo.

Output

Imprime una sola línea con un entero no negativo — la respuesta al problema.

Examples

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

Note

En el primer caso de prueba hay 3 pares: $(1, 2)$, $(2, 3)$ y $(1, 3)$.

En el segundo caso de prueba hay 6 pares: $(4, 5)$, $(4, 6)$, $(5, 6)$, $(4, 4)$, $(5, 4)$ y $(5, 5)$.

En el tercer caso de prueba hay 4 pares: $(7, 8)$, $(7, 7)$, $(8, 8)$ y $(8, 7)$.

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 ama los concursos de TV. Su concurso preferido se llama "Figuras Escondidas". En este concurso hay diferentes tipos de retos que consisten en encontrar la cantidad de figuras escondidas que hay entre un montón de puntos. La única diferencia entre los retos está en el tipo de figura que hay que buscar, estas pueden ser cuadrados, rectángulos, triángulos, etc.

El tipo de reto más difícil del concurso es el de buscar triángulos, pero es también el que más dinero otorga. Luisa ha llamado dos veces intentando ganar el premio mayor de este reto pero no ha podido acertar a la respuesta correcta. Por eso, ella quiere estar preparada esta vez.

Luisa te pide que elabores un programa que, dado un conjunto de puntos, pueda encontrar la cantidad de *triángulos válidos* diferentes que se puedan formar. Un triangulo es válido si sus vértices están entre los puntos dados y su área es mayor a 0. Un triángulo es diferente a otro si al menos uno de sus vértices es diferente a cualquiera del otro.

Input

La entrada comienza con un entero T indicando el número de casos de prueba. Para cada caso de prueba, la primera línea consiste en un entero n $(3 \leq n \leq 100)$ indicando el número de puntos que se muestran en pantalla.

Las siguientes n líneas contendrán dos enteros x e y $(-100 \leq x, y \leq 100)$ indicando las coordenadas (x, y) de los puntos mostrados en la pantalla sobre un plano cartesiano. Se garantiza que no hay dos puntos en las mismas coordenadas.

Output

Por cada caso imprime una línea con un número entero no negativo indicando la cantidad de triángulos que Luisa debe responder para ganar el premio mayor.

Examples

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)`

Cuando Crisel y Carlos fueron a Buenos Aires, Argentina, ellos aprovecharon su estadía allí para visitar lugares maravillosos como El Obelisco, Caminito, El Parque de la Memoria, entre muchos otros.

Para evitar perderse, ellos dibujaron un mapa con los n lugares que querían visitar y etiquetaron cada lugar con la letra inicial de su respectivo nombre. Además, dibujaron $n-1$ carreteras de doble vía conectando pares de lugares de tal manera que fuese posible moverse entre dos lugares cualesquiera usando solo estas carreteras (quizás, pasando por algunos lugares intermedios).

Hoy en día, ellos están aburridos en sus casas debido a la cuarentena. Sin embargo, hoy Crisel encontró el mapa que ellos habían hecho y se le ocurrió un juego de rondas, así que llamó a Carlos para contarle su idea.

Hay dos tipos de rondas, en una ronda de tipo 1, cada jugador elije uno de los n lugares del mapa. Luego, forman una cadena que consta de las etiquetas en cada uno de los lugares por los que deben pasar para ir del lugar elegido por Crisel al lugar elegido por Carlos (incluyendo los lugares elegidos por ambos jugadores). Si la palabra resultante es un **palíndromo**, ¡entonces se dice que es un *viaje palindrómico*! El primero en decir si es o no un viaje palindrómico gana la ronda.

En una ronda de tipo 2 Crisel puede cambiar la etiqueta de un lugar en el mapa con otra letra elegida aleatoriamente.

Ahora que también tienes el mapa ¿Podrías decir, para cada ronda de tipo 1, si es un viaje palindrómico?

Input

La primera línea contiene dos enteros n y r ($2 \leq n \leq 10^5$, $1 \leq r \leq 10^5$) — la cantidad de lugares en el mapa y la cantidad de rondas, respectivamente.

La siguiente línea contiene n letras minúsculas en inglés, que corresponden a las etiquetas iniciales de cada lugar (desde el lugar 1 hasta el lugar n).

Cada una de las siguientes $n-1$ líneas contiene dos enteros u y v ($1 \leq u, v \leq n$, $u \neq v$), indicando que hay una carretera que conecta los lugares u y v.

Las siguientes r líneas contienen la descripción de cada ronda:

- Una ronda de tipo 1 comienza con un 1 seguido de dos enteros u y v ($1 \leq u, v \leq n$) indicando que Crisel eligió el lugar u y Carlos el lugar v.
- Una ronda de tipo 2 comienza con un 2 seguido de un entero x y un carácter c ($1 \leq x \leq n$, c es una letra minúscula en inglés) indicando que Crisel estableció la etiqueta del lugar x como la letra c.

Las rondas se dan en el orden en que ocurren. Se garantiza que la última ronda es de tipo 1.

Output

Para cada ronda de tipo 1, imprime "Yes" si es un viaje palindrómico, si no imprime "No" (sin las comillas).

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

En el primer caso, las palabras escritas en cada ronda de tipo 1 son respectivamente:

- "fus" o "suf".
- "sus".
- "ff".

Una palabra es llamada un **palíndromo** si se lee igual de izquierda a derecha y de derecha a izquierda. Por ejemplo, "abacaba", "cc" y "m" son palíndromos, pero "go", "reloaded" y "equipo" no lo son.

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

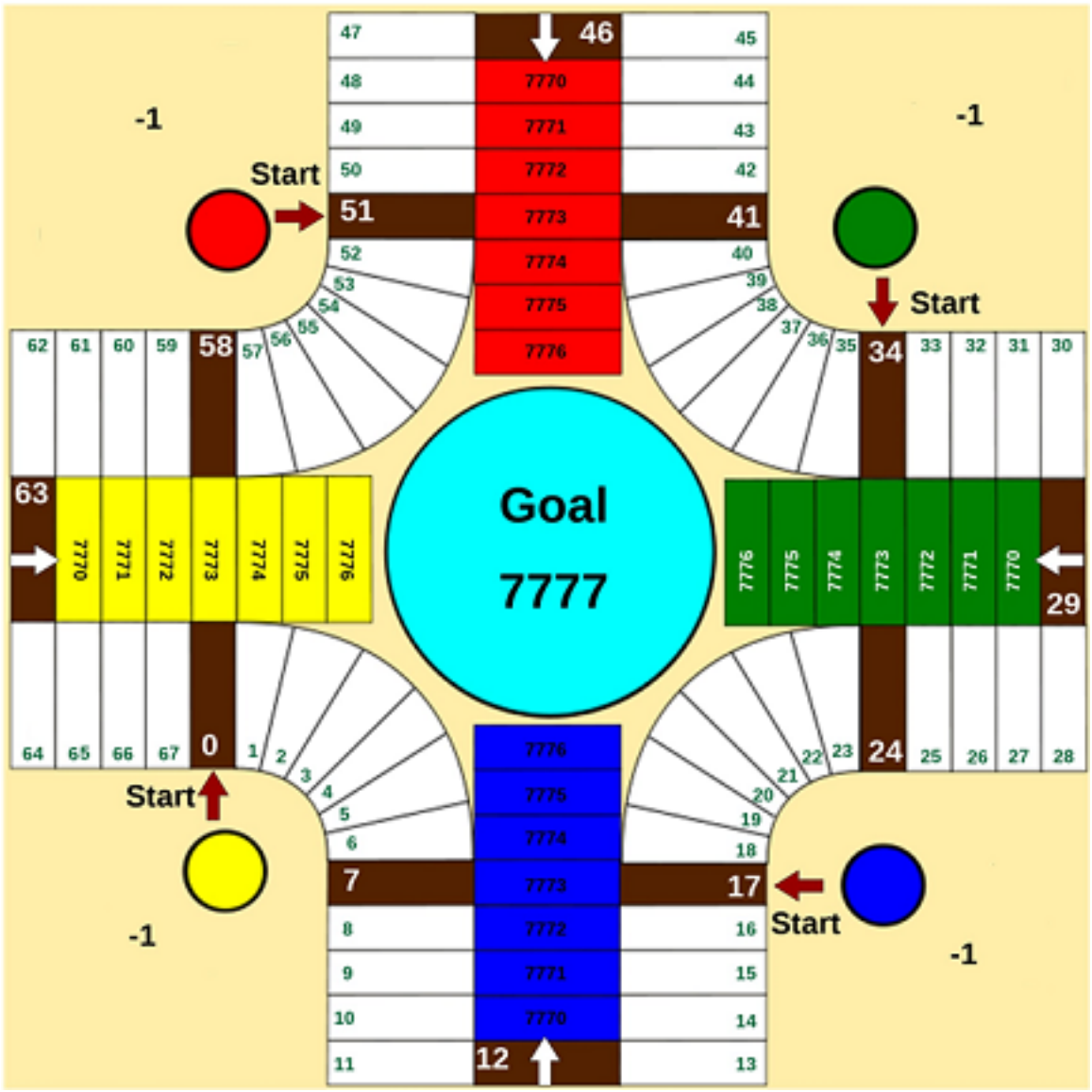
Input	Output
10 1 1 5 CARLOS 1 3 4 MANUEL 1 4 4 ANDRES 1 6 1 GABRIEL 2 9 1 13 2 YEFRI 2 20 1 24 1 STEVEN 2 30 2 50	6 9 3 GABRIEL 13 20 7 YEFRI 24 30 6 STEVEN 3 50 47 MANUEL

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)`

El parqués es un juego muy conocido que todos saben jugar. Consiste en un tablero, dos dados y una o más fichas por cada jugador. En este caso se usará solo una ficha. Un tablero de parqués puede ser para 4 jugadores, para 5 jugadores, para 6 jugadores, etc. El tablero para 4 jugadores es el tablero base y su diseño se puede extender para cualquier cantidad de jugadores. La siguiente figura muestra el diseño para un tablero de 4 jugadores y la codificación propuesta para representar el tablero en un computador.



El tablero tiene cuatro zonas:

1. La primera zona es el inicio del juego y cada jugador tiene su propia zona privada. Está codificada

con −1. Para salir de esta zona el jugador debe obtener un número doble al lanzar los dados (11, 22, 33, 44, 55 o 66). Y saldrá a la casilla marcada con una flecha como su inicio.

2. La segunda zona es la meta del juego, el primer jugador que llegue aquí gana. Está codificada con 7777.

3. La tercera zona es general y tiene 17 casillas por cada jugador. Un tablero para 4 jugadores tiene 68 casillas en esta zona, para 5 jugadores tiene 85 casillas, para 10 jugadores tiene 170 casillas y así sucesivamente. En la figura anterior, esta zona corresponde a las casillas de color blanco y marrón. Está codificada desde 0 hasta $17 * K - 1$ (para K jugadores). Las casillas de color blanco solo permiten que haya una ficha a la vez por casilla. Es decir, que si hay colisión de dos fichas en una casilla de color blanco, la ficha que llega hace que la otra regrese a la primera zona e inicie de nuevo el juego. Las casillas de color marrón son seguras y permiten que haya varias fichas a la vez por casilla.

4. La cuarta zona es segura y cada jugador tiene su propia zona privada con un color personalizado. Está codificada con números desde 7770 hasta 7776: el camino a la meta.

Las reglas del juego son muy sencillas:

1. El juego es por turnos. Un jugador en su turno debe lanzar los dados y mover su ficha según las reglas siguientes. Siempre inicia el jugador que está junto a la posición cero (0). Luego, los turnos siguen el orden en que está enumerada la tercera zona.

2. Para salir de la primera zona el jugador debe obtener un número doble al lanzar los dados (11, 22, 33, 44, 55 o 66).

3. Si un jugador lanza los dados y saca un número doble, luego de hacer su movimiento, debe volver a lanzar los dados como si tuviese un turno nuevo.

4. Si el jugador se encuentra en la tercera zona, avanza tantas casillas como la suma de los valores obtenidos al lanzar los dados.

5. Si el jugador se encuentra en la cuarta zona y la suma de los valores obtenidos al lanzar los dados es mayor que lo necesario para llegar a la meta entonces debe avanzar y regresar. Por ejemplo, si el jugador se encuentra en 7770 y al lanzar los dados obtiene un 65 entonces debe avanzar 11 casillas, por lo cual llega a la meta 7777 y regresa a 7773.

6. Si el jugador se encuentra en la cuarta zona y la suma de los valores obtenidos al lanzar los dados es igual a lo necesario para llegar a la meta entonces el jugador gana y el juego finaliza. El jugador que gana queda ubicado en la posición 7777.

7. Mientras no haya un ganador, cada jugador continua jugando y sigue las reglas del juego.

Se quiere desarrollar un programa de computador que reciba una lista de lanzamientos de dados para un juego con una cantidad K de jugadores. El programa debe informar si hubo un ganador, cuál fue el ganador y la secuencia de casillas del tablero ocupadas por cada jugador después de cada lanzamiento de dados.

Input

La entrada consiste de múltiples casos de prueba. Cada caso de prueba consta de tres líneas:

La primera línea contiene un número entero K $(4 \leq K < 100)$ que indica la cantidad de jugadores.

La segunda línea contiene un número entero N ($K \leq N < 10^5$) que indica la cantidad de lanzamientos de dados.

Y la tercera línea contiene N números de dos dígitos separados por un solo espacio, que indican cada uno de los lanzamientos de dados. Cada uno de los dígitos corresponde al resultado de un dado. Se garantiza que solo se usan los dígitos 1, 2, 3, 4, 5 y 6. El final de los casos de prueba es el fin de archivo (EOF).

Output

Para cada caso de prueba deben imprimirse $K + 1$ líneas.

Las primeras K líneas corresponden a los números de las casillas ocupadas por cada jugador durante el juego, incluyendo la zona de inicio del juego -1 y las repeticiones de casillas que se puedan presentar, usando el siguiente formato:

Cada línea inicia con la palabra "`Player`" seguida por un espacio y el número del jugador (de 1 a K). Luego otro espacio, el símbolo "`=`", otro espacio y finalmente una secuencia de números encerrados en llaves, separados por coma y sin espacios entre ellos:

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

La última línea (línea $K + 1$) informa el ganador "`The winner is Player W`" (donde W es el número del jugador ganador) o informa que aún no hay ganador "`There is no winner yet`". En caso de haber ganador, se garantiza que ese jugador gana la partida con el lanzamiento de dados N.

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)`

El Lazy Team es un equipo que odia los problemas con enunciados largos. Sin muchas aspiraciones y con mucha pereza deciden solo leer e intentar los problemas con enunciados cortos (enunciados con menos de una página). Aunque no se ven como los futuros campeones regionales, tampoco quieren quedar en blanco en la competencia.

Una competencia consta de problemas de tres dificultades diferentes (fácil, medio y difícil). Denotemos el número de problemas fáciles, medios y difíciles como a, b y c respectivamente. Por tanto, hay un total de $a+b+c$ problemas. Además, denotemos el número de problemas con enunciados cortos como n $(0 \leq n \leq a+b+c)$.

Antes del concurso, el equipo decidió usar su computador para hacer algunas simulaciones. Ellos quieren saber cuál es la probabilidad de recibir al menos k problemas fáciles entre los n problemas con enunciados cortos de la competencia. Ellos pueden asumir que la probabilidad de que un problema tenga un enunciado corto es la misma para todos los problemas.

Como son tan perezosos, también les da pereza hacer este cálculo, ¿puedes ayudarles?

Input

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

Cada una de las siguientes T líneas incluye una simulación. Una simulación consiste de cinco números enteros no negativos a, b, c $(1 \leq a+b+c \leq 16)$, n $(0 \leq n \leq a+b+c)$ y k $(0 \leq k \leq \min(a,n))$ que corresponden a la cantidad de problemas fáciles, medios y difíciles, la cantidad de problemas con enunciados cortos y la cantidad de problemas fáciles que esperan elegir, respectivamente.

Output

Para cada simulación, imprime una línea con la respuesta al problema redondeada a cuatro dígitos después del punto decimal.

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)`

Este problema es el mismo que "Once Again: Minesweeper App". La única diferencia entre las dos versiones está en los limites sobre el tamaño del tablero.

Natalia y Fernando trabajan en una prestigiosa empresa llamada WBOSS. Hace unos días, su jefe les asignó un nuevo proyecto: Desarrollar una aplicación móvil del famoso juego "Buscaminas".

Cada nivel del juego será un tablero rectangular con n filas y m columnas. Cada una de las diferentes $n \times m$ celdas del tablero puede contener un número o una mina. Las celdas que contendrán números ya están definidas por el cliente pero las celdas que contendrán minas no.

El juego debe ser fácil, por lo que el tablero de cada nivel debe tener la mínima cantidad de minas posible y debe ser un *tablero válido*. Un tablero T es válido si para cada celda en la posición (i, j) que contiene un número ($1 \leq i \leq n$; $1 \leq j \leq m$), hay exactamente $T_{i,j}$ minas en sus celdas adyacentes (en las ocho direcciones posibles). Denotamos $T_{i,j}$ como el número que está en la celda de la posición (i, j) del tablero T.

Por ultimo, el juego debe tener la mayor cantidad de niveles y todos deben ser diferentes. Dos niveles son diferentes si hay al menos una posición que contiene una mina en uno de los tableros pero en el otro no.

Natalia renunció a WBOSS por una oferta muy bien pagada, por lo que el proyecto ahora está en manos de Fernando. ¿Podrías ayudarlo a calcular la mínima cantidad de minas que debe tener un tablero para ser considerado válido? Además de eso, ¿podrías determinar cuántos niveles diferentes puede hacer con esta cantidad de minas?

Input

La primera línea contiene dos enteros n y m ($1 \leq n * m \leq 20$) que indican la cantidad de filas y columnas de cada tablero T en el juego, respectivamente.

Luego siguen n líneas cada una con m caracteres. Cada carácter es un "`.`" o un dígito d ($1 \leq d \leq 8$). Si el j-ésimo carácter de la i-ésima línea es "`.`", entonces la celda de la posición (i, j) está vacía. De lo contrario, el dígito d representa el valor de $T_{i,j}$.

Output

La salida consiste de una sola línea. Si es imposible desarrollar el juego imprime "`0`" (sin las comillas).

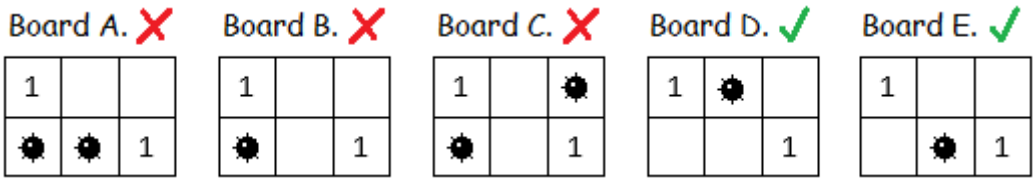
De lo contrario, imprime dos enteros a y b — la cantidad de niveles y la cantidad de minas, respectivamente.

Examples

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

Note

Para el primer caso de prueba, el siguiente gráfico muestra cinco tableros de ejemplo:



El tablero A no es válido porque la celda $A_{1,1} = 1$, pero hay 2 minas en sus celdas adyacentes.

El tablero B no es válido porque la celda $B_{2,3} = 1$, pero no hay minas en sus celdas adyacentes.

El Tablero C es un tablero válido pero no tiene la mínima cantidad de minas posible.

Los tableros D y E son válidos y tienen la mínima cantidad de minas posible. Además, son diferentes.

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 está entrenando para dominar el antiguo y sagrado arte de las "Maratones de Programación". Para esto, cuenta con un maestro llamado JCB al que admira mucho y que le da un plan de entrenamiento para prepararse. El entrenamiento de JCB es similar al entrenamiento del señor Miyagi a Daniel San, que incluía un conjunto de tareas como encerar el auto o pintar la cerca. Ahora JCB le ha asignado una misión a Manuel que ni siquiera el señor Miyagi se hubiera atrevido a asignar. ¿Podrá Manuel completar esta misión a tiempo y así dominar el siguiente nivel de las "Maratones de Programación"?

La misión comienza con un arreglo de N strings y un pergamino sagrado que Manuel debe encontrar en lo mas profundo del Himalaya. Una vez obtenido, Manuel debe completar una serie de Q tareas escritas en el pergamino sagrado. Cada una de estas tareas consta de un intervalo $[L, R]$. Sin embargo, el pergamino solo puede ser leído por quienes lo merezcan. Por esto, el pergamino tiene dos valores A y B por cada tarea y Manuel debe usarlos para calcular el intervalo real $[L, R]$ de la siguiente manera:

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

En caso de que $L > R$, Manuel puede simplemente intercambiar los valores L y R para que la condición $L \leq R$ siempre se cumpla. $Last$ representa la respuesta de la tarea anterior y se puede asumir que es 0 antes de la primera tarea.

Manuel pregunta a su maestro JCB cuál es la tarea que debe completar con este intervalo. A esto JCB responde lo siguiente de manera muy abstracta y confusa:

"La respuesta de cada tarea puede encontrarse en el fondo de tu corazón. Mira dentro de ti y luego calcula la cantidad de substrings distintos que hay en el intervalo del arreglo desde la posición L hasta R (incluyendo tanto L como R)."

Manuel sigue confundido porque el maestro habla de una forma muy extraña, así que acude al Oráculo para obtener mas información. Manuel le entrega al Oráculo unas monedas de oro y el Oráculo muy amablemente le muestra a Manuel una serie de ejemplos sobre las tareas que debe realizar y una explicación de estos.

Input

La entrada inicia con dos valores N ($1 \leq N \leq 5000$) y Q ($1 \leq Q \leq 10^6$) que representan el tamaño del arreglo A y el numero de tareas deben completarse respectivamente.

Luego recibirá N strings A_i que representan los elementos del arreglo. Puede asumir que la suma de las longitudes de todos los strings A_i es ≤ 5000 (es decir, $\sum_{i=1}^{N} |A_i| \leq 5000$). Cada string consiste únicamente de letras minúsculas en Ingles ($a - z$).

Después de esto, recibirá Q líneas, cada una incluyendo dos números A ($1 \leq A \leq N$) y B ($1 \leq B \leq N$) que representan cada una de las tareas que Manuel debe completar.

Output

Imprime Q líneas cada una con un entero V_i que representa la respuesta para la i-ésima tarea.

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

Note

Los substrings de cada elemento del array son los siguientes:

banana → {a, an, ana, anan, anana, b, ba, ban, bana, banan, banana, n, na, nan, nana}

ana → {a, an, ana, n, na}

j → {j}

Cada tarea se resuelve de la siguiente manera:

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

subarray → {banana}

15

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

subarray → {banana, ana} (todos los substrings de ana son substrings de banana)

15

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

subarray → {banana, ana, j} (todos los substrings de ana son substrings de banana)

16

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

subarray → {ana}

5

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

subarray → {ana, j}

6

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

subarray → {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)`

Este problema es el mismo que "Minesweeper App". La única diferencia entre las dos versiones está en los limites sobre el tamaño del tablero.

Natalia y Fernando trabajan en una prestigiosa empresa llamada WBOSS. Hace unos días, su jefe les asignó un nuevo proyecto: Desarrollar una aplicación móvil del famoso juego "Buscaminas".

Cada nivel del juego será un tablero rectangular con n filas y m columnas. Cada una de las diferentes $n \times m$ celdas del tablero puede contener un número o una mina. Las celdas que contendrán números ya están definidas por el cliente pero las celdas que contendrán minas no.

El juego debe ser fácil, por lo que el tablero de cada nivel debe tener la mínima cantidad de minas posible y debe ser un *tablero válido*. Un tablero T es válido si para cada celda en la posición (i, j) que contiene un número ($1 \leq i \leq n$; $1 \leq j \leq m$), hay exactamente $T_{i,j}$ minas en sus celdas adyacentes (en las ocho direcciones posibles). Denotamos $T_{i,j}$ como el número que está en la celda de la posición (i, j) del tablero T.

Por ultimo, el juego debe tener la mayor cantidad de niveles y todos deben ser diferentes. Dos niveles son diferentes si hay al menos una posición que contiene una mina en uno de los tableros pero en el otro no.

Natalia renunció a WBOSS por una oferta muy bien pagada, por lo que el proyecto ahora está en manos de Fernando. ¿Podrías ayudarlo a calcular la mínima cantidad de minas que debe tener un tablero para ser considerado válido? Además de eso, ¿podrías determinar cuántos niveles diferentes puede hacer con esta cantidad de minas?

Input

La primera línea contiene dos enteros n y m ($1 \leq n * m \leq 60$) que indican la cantidad de filas y columnas de cada tablero T en el juego, respectivamente.

Luego siguen n líneas cada una con m caracteres. Cada carácter es un "`.`" o un dígito d ($1 \leq d \leq 8$). Si el j-ésimo carácter de la i-ésima línea es "`.`", entonces la celda de la posición (i, j) está vacía. De lo contrario, el dígito d representa el valor de $T_{i,j}$.

Output

La salida consiste de una sola línea. Si es imposible desarrollar el juego imprime "`0`" (sin las comillas).

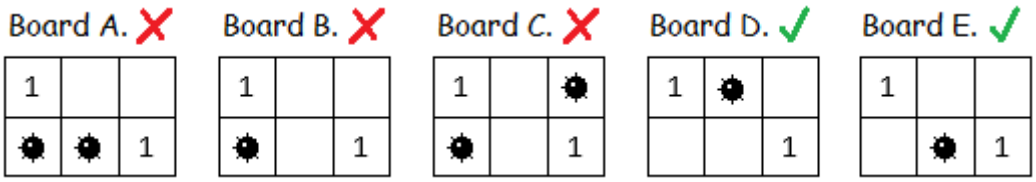
De lo contrario, imprime dos enteros a y b — la cantidad de niveles y la cantidad de minas, respectivamente.

Examples

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

Note

Para el primer caso de prueba, el siguiente gráfico muestra cinco tableros de ejemplo:



El tablero A no es válido porque la celda $A_{1,1} = 1$, pero hay 2 minas en sus celdas adyacentes.

El tablero B no es válido porque la celda $B_{2,3} = 1$, pero no hay minas en sus celdas adyacentes.

El Tablero C es un tablero válido pero no tiene la mínima cantidad de minas posible.

Los tableros D y E son válidos y tienen la mínima cantidad de minas posible. Además, son diferentes.