Maratón de Programación 2021

Universidad Francisco de Paula Santander
Vigilada Mineducación

Programa de Ingeniería de Sistemas
Acreditado de Alta Calidad
"Educación y Tecnología con Compromiso Social"

SILUX

RED DE PROGRAMACIÓN COMPETITIVA RPC

Hay un lugar para tí en la historia de la Programación Competitiva.

Grupo de Estudio
Programación Competitiva

FECHA: Sábado 11 de Diciembre de 2021

HORA: 12:00 Medio día hasta las 7:00 p.m.

LUGAR: Virtual

C++ python Java

Contents

1 Séptima Maratón de Programación UFPS - 2021

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.

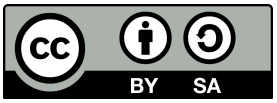
Los pioneros de dicho proyecto fueron dos estudiantes, quienes ya se graduaron y dejaron como herencia una gran motivación. Ahora siguen colaborando con el Semillero que es liderado directamente por los estudiantes, quienes ingresan desde primer semestre con el entusiasmo que trae el sueño de llegar 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 2021 nos complace presentar un conjunto de trece (13) problemas inéditos, los cuales fueron escritos por estudiantes, profesores y graduados de varias universidades. Por la pandemia del COVID19, este es el segundo año virtual, con lo cual damos un ejemplo de confianza, transparencia y visión en estas competencias.

Reconocimiento y agradecimiento especial al equipo de trabajo:

- Eloy Pérez Torres - Profesor Universidad de la Habana
- Eddy Ramírez Jiménez - Profesor UNA & TEC Costa Rica
- Hugo Humberto Morales Peña - Profesor Universidad Tecnológica de Pereira, Colombia
- Juan Manuel Reyes - Profesor Icesi Cali
- Milton Jesús Vera Contreras - Profesor UFPS (desde Cúcuta)
- Gerson Yesid Lázaro - Graduado UFPS (desde Bogotá)
- Angie Melissa Delgado - Graduado UFPS (desde Cali)
- José Manuel Salazar Meza - Graduado UFPS (desde Cúcuta)
- Wilmer Emiro Castrillón Calderón - Graduado Universidad de la Amazonía, Colombia
- Juan José Ortiz Plaza - Estudiante Universidad de la Amazonía, Colombia
- Equipo Red de Programación Competitiva (Diana Espinosa, Fabio Avellaneda, Johany 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 habilidades hard como habilidades soft.

Los integrantes del grupo de estudio han participado en la Maratón Nacional de Programación desde el año 2014, logrando por 7 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 o basename 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 tu solución es correcta, en unos momentos la plataforma te lo indicará con el texto "YES". En caso contrario, obtendrá un mensaje de error.

4 Reglas

1. Se permite el uso de computador personal a los 3 integrantes del equipo al mismo tiempo.
2. No se permite el uso de internet durante la competencia con el objetivo de buscar material que ayude a resolver los problemas (a excepción del notebook del equipo).
3. No se permite copiar y pegar código desde fuentes disponibles en Internet (a excepción del notebook del equipo).
4. No se permite la comunicación entre miembros de equipos diferentes. Cada integrante solo puede comunicarse con sus dos compañeros de equipo o con los jueces por medio de las clarificaciones en la plataforma.

5. Se permite todo tipo de material impreso (libros, fotocopias, apuntes, cuadernos, guías) que el equipo desee utilizar.

6. 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 siguiente).

7. 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).

5 Lista de Problemas

A continuación la lista de los trece (13) problemas a resolver. Un primer reto y logro es resolver alguno de los problemas. Un segundo reto es lograr resolver cualquiera de los problemas más rápido que todos los demás. Un tercer reto es lograr resolver todos los problemas. Un cuarto reto es lograr unicarse en el top 3 de la competencia. Y un quinto reto es lograr ubicarse en el top 5 o ser el mejor equipo novato ;)

Pero recuerda que solo al competir ya se es ganador ;)

Nota: A continuación, en la siguiente página, encontrará el detalle de tiempos en C++, Java y Python, conforme a las pruebas realizadas por el equipo de trabajo de RPC y UFPS, usando el servidor de Boca de RPC y UFPS. Los tiempos de Python no se pueden garantizar al 100%.

Tiempo en segundos (servidor Boca Red de Programación Competitiva - Maratón UFPS 2021

Letra	fullname	C y C++		Java		Python *	
		1 test case	All test case	1 test case	All test case	1 test case	All test case
A	A Twin Prime Number	1	3	2	5	2	5
B	Beginner: this is your challenge	1	3	2	5	2	5
C	Carlos's Game	1	6	1	45	1	7
D	Dangie's Function	1	1	2	5	2	5
E	Enjoys playing with glass vases	1	2	1	7	1	7
F	Vicsek's Fractal	1	2	1	2	1	2
G	Gardener	1	60	5	90	5	90
H	Harry Potter and the Portkey Problem	1	3	6	30	6	30
I	Ivan Looks for a Console	1	1	2	5	2	5
J	Jum Hugo sabe	1	1	2	5	2	5
K	Kidnapped	1	1	2	2	2	2
L	Lucky is a Crazy Dog	1	1	2	5	2	5
M	Mixture	1	1	5	5	5	5

* En python no se pueden garantizar los tiempos al 100%

Problem A. A Twin Prime Number

Source file name:	Atwin.c, Atwin.cpp, Atwin.java, Atwin.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Juan José Ortiz Plaza - U. de la Amazonía (Student)`

The Professor Elio Sabio likes twin prime numbers. A twin prime is a prime number that is two more or two less than another prime number.

On the day of the final exam of the course the Professor decided to give different problems to his students about twin prime numbers and you were given the following one:

Given a number n, find the prime twin number closest to n, if there are two numbers that are at the same distance from n you should choose the larger one.

Input

The first input line contains a positive integer $1 \leq t \leq 10^5$, corresponding to the number of test cases, then there will be t lines each containing a positive integer $1 \leq n \leq 10^6$.

Output

For each input line, you will need to generate an output line that contains the closest twin prime to n.

Examples

Input	Output
4 1 2 5 12	3 3 5 13

Problem B. Beginner: this is your challenge

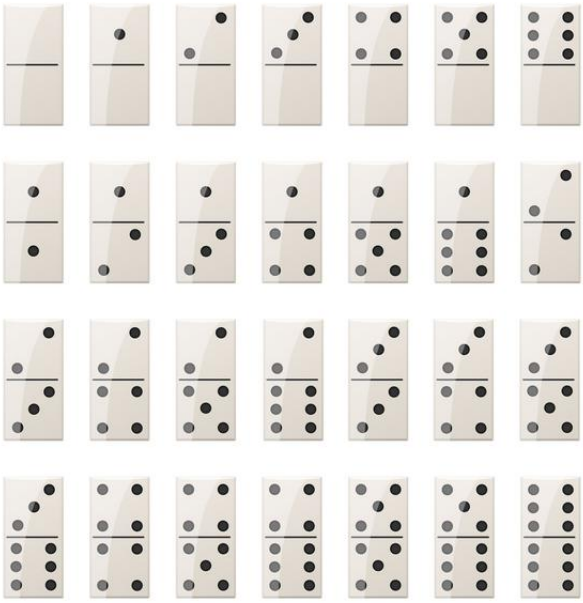
Source file name:	Beginner.c, Beginner.cpp, Beginner.java, Beginner.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Milton Jesús Vera Contreras - UFPS (Professor)`

The purpose of this challenge is than first semester students to win the award for best beginner team. So that they are motivated to continue leading competitive programming at UFPS.

The game of Dominoes is ancient, popular, simple and mysterious! There is a lot of interesting math in this game ...

The rules of the game can be summarized as follows:

- It is a game based on twenty-eight (28) slabs, each of which has two sides and on each side there are p points ($0 \leq p \leq 6$), without repetitions, as in the figure:



- The tiles can be distributed between two (2), four (4) or more players, normally the maximum number of players is four (4).
- Each tile can be represented by one or two numbers, depending on the order of its sides. For example,: tile 15 is the same tile 51, tile 10 is the same tile 1 (zero to the left is ignored) and tile 34 is the same tile 43. Tiles with doubles only have one number that represents them : 0, 11, 22, 33, 44, 55 and 66.
- The game always starts with a double tile, normally double six 66, but any other can be agreed, such as double five 55 or double zero 00.
- The tiles must be placed on a table, adjacently, matching the two sides that have the same number of points p. For example, with two players, you can start with 66, then 65, then 52, and so on. This adjacency rule is mandatory to consider that the game is valid.

- The game is continued until the tiles are finished or until the game is closed, which will be explained later.

Any arrangement of tiles, of any quantity k can be represented by a number n, usually very large, with the following equation, which we will call the **General Domino Equation**:

$$n=\sum_{i=0}^{k} f_i * 10^{2i}$$
$$f_i \in \{0,1,2,3,4,5,6,$$
$$10,11,12,13,14,15,16,$$
$$20,21,22,23,24,25,26,$$
$$30,31,32,33,34,35,36,$$
$$40,41,42,43,44,45,46,$$
$$50,51,52,53,54,55,56,$$
$$60,61,62,63,64,65,66\}$$
$$\forall i, j\, i \neq j \rightarrow f_i \neq f_j$$
$$\forall i > 0 \wedge i < k\ remainder(f_i, 10) = quotient(f_{i-1}, 10)$$

For example, the following arrangement of ten ($k = 10$) tiles corresponds to the number $54,422,555,511,665,500,333$ fifty-four trillion four hundred twenty-two thousand five hundred fifty-five billion five hundred eleven thousand six hundred sixty-five million five hundred thousand three hundred thirty-three (in Colombia):

i	9		8		7		6		5		4		3		2		1		0	
f_i	5	4	4	2	2	5	5	5	5	1	1	6	6	5	5	0	0	3	3	3

A particular case of dominoes is the **Perfectly Closed Game**. This case consists in that the two sides of the game have the same amount of points p and all the seven (7) tiles that have that amount of points at their sides are already on the table, they have already been played. For example, these two scenarios correspond to perfectly closed games with $p = 6$.

i	9		8		7		6		5		4		3		2		1		0	
f_i	6	1	1	4	4	6	6	6	6	5	5	3	3	6	6	0	0	2	2	6

i	9		8		7		6		5		4		3		2		1		0	
f_i	6	2	2	0	0	6	6	3	3	5	5	6	6	6	6	4	4	1	1	6

A Perfectly Closed Game has a size of k, which corresponds to the amount of tiles that have been played. We are interested in the cases of $k = 10$ which are the smallest and simplest and of which the teacher has

calculated more than ten thousand 10^4 cases for this challenge, all similar to the previous two. For these ten tile scenarios:

- You can easily identify if the game is perfectly closed.
- If that game is represented by a number n with the equation previously described, it is easy to write a computer program that determines if the game is perfectly closed.
- If some of its digits are disordered to that number n, that is, a permutation of the number n is applied, a second number m is obtained, which does not always represent a valid game, it does not always satisfy the equation domino general. The two example scenarios above are mutually permutation and represent valid games, they satisfy the equation. And the following figure shows two permutations that are not valid games, they do not satisfy the equation.

i	9		8		7		6		5		4		3		2		1		0	
f_i	2	1	3	6	4	6	6	6	6	4	0	3	1	6	6	5	0	2	6	5

i	9		8		7		6		5		4		3		2		1		0	
f_i	0	2	2	0	6	1	6	6	3	5	5	6	6	6	6	4	3	6	1	4

Do you want to win the prizes of the best beginner team?
Solve this challenge, you can do it!

Input

Several test cases. Each test case consists of a twenty (20) digit number. Each digit d satisfies $0 \leq d \leq 6$.

Output

For each test case, print a line with three messages YES or NO, separated by a blank space, depending on whether or not each of the following conditions are satisfied:

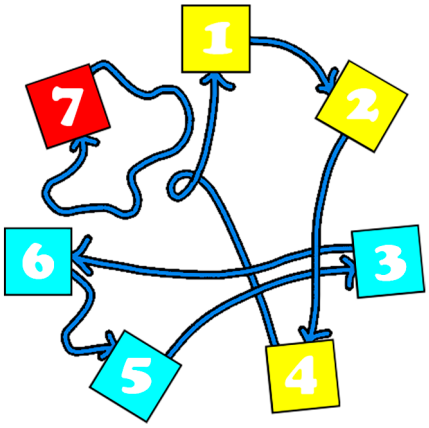
- The number m satisfies the **General Domino Equation** and therefore corresponds to a valid domino game.
- The number m corresponds to a permutation of number n and n represents a **Perfectly Closed Game** of size ten $k = 10$.
- The number m corresponds to a **Perfectly Closed Game** of size ten $k = 10$.

Examples

Input	Output
61144666655336600226	YES YES YES
26611446666553366002	YES YES NO
54422555511665500333	YES NO NO
21364666640316650265	NO YES NO
45425255151665503330	NO NO NO

Problem C. Carlos's Game

Source file name:	Cgame.c, Cgame.cpp, Cgame.java, Cgame.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`José Manuel Salazar Meza (Graduate)`



Carlos is one of the many students for whom the change from in person classes to virtual classes (in an improvised way) turned his college student days into something not as fun as they used to be. To pass the time Carlos, who is very intelligent and creative, invented a game called "Carlos's Game" and it consists of the following:

First, Carlos chooses a positive integer n and draws a board with n squares numbered from 1 to n arranged in a circular way like a clock (the square 2 is in front of the 1, the square 3 is in front of the 2, square 1 is in front of the n, etc). Then, he draws one-way roads from each square i to the square that is i positions ahead on the board.

Once the board is ready, Carlos chooses two random squares x and y from the board and places a token on each square. Finally, he chooses two positive integers a and b that indicate how many squares the tokens move each turn (following the one-way roads), respectively.

The challenge of "Carlos's Game" is to find out how many turns must pass for the two tokens to meet for the first time on the same square or to prove that the two pieces will never meet.

Carlos has been playing for a long time and he's already an expert. He knows that if $x = n$ or $y = n$, it's very easy to solve the challenge, so he now discards those cases. He also thinks that it isn't fun to choose small values for n in such a general way, so now he only plays cases where $n \geq 3$ and is a **prime number**. Also, n can be very large.

Currently playing "Carlos's Game" is really hard, so hard that Carlos hasn't even returned to his virtual classes and has been trying to solve the challenge for days. Will you be able to solve it before him?

Input

The input consists of a single line containing five integers n, x, a, y and b ($3 \leq n \leq 10^{10}$ and **is prime**, $1 \leq x, y < n$, $1 \leq a, b \leq 10^{10}$) that indicate the integers chosen by Carlos described above.

Output

The output consists of a single line. If the two tokens never meet, print -1. Otherwise, print two integers k and z indicating how many turns must pass for the two tokens to meet for the first time on the same square and what that square will be, respectively.

The answer is guaranteed to fit into a 64-bit integer.

Examples

Input	Output
5 1 2 2 3	3 4
7 4 2 3 1	-1
100000007 67108864 2 1048576 3	6 77887708

Note

Note

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

Problem D. Dangie's Function

Source file name:	Dangie.c, Dangie.cpp, Dangie.java, Dangie.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia (Professor)`

Typically in a first-year programming course in an engineering or computer science academic program, students are taught to build functions that make use of the doubly nested loop structure such as the following:

```
unsigned long long DangiesFunction(int n)
{
    int i, j, k;
    unsigned long long result = 0;

    for(i = 1; i <= n - 1; i++)
    {
        for(j = i + 1; j <= n; j++)
        {
            for(k = 1; k <= j; k++)
            {
                result += 1;
            }
        }
    }

    return result;
}
```

The running-time function of the previous algorithm belongs to $O(n^3)$, with a value of $n = 10^6$ the total number of operations performed by the algorithm, in order to give the result would require 10^{18} steps. As a competitor of the different phases of the ICPC, you know that a solution that performs that amount of steps, will obtain a verdict of **Time Limit Exceeded** in competition, for this reason you are asked to propose a solution that has a running time as close to $O(1)$ as possible, regardless of the size of n, in which you must make use of all your experience as a competitor of ICPC!

Input

The input begins with a positive integer t ($1 \leq t \leq 10^6$), which represents the total number of test cases. Then t lines are presented, each one containing a positive integer n ($1 \leq n \leq 10^6$) for which the result of the **Dangie's Function** must be calculated.

Output

The output must contain t lines, each containing a long positive integer as a result of the **Dangie's Function**.

Examples

Input	Output
7 1 5 25 52 833 5234 1000000	0 40 5200 46852 192669568 47794715890 333333333333000000

Problem E. Enjoys playing with glass vases

Source file name:	Enjoy.c, Enjoy.cpp, Enjoy.java, Enjoy.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Eloy Pérez Torres - Universidad de la Habana (Professor)`

Magus, the sorcerer, enjoys playing with glass vases. In this occasion he has two vases, A and B, with capacities of Ca and Cb liters of water respectively. Initially both vases are empty. Magus can completely fill any vase with water, completely empty any of them or even pour water from one vase to the other one avoiding water loss. He has been filling, emptying and pouring water into the two vases for at least two hours, so he lost count on how many different water configurations over A and B he had seen so far. He demands you to find the total amount of different water configurations in vases A and B if the only available operations are: completely fill a glass vase, completely empty a glass vase and pour water from one vase to another without water loss.

Input

The first and only line of input contains two positive integer numbers Ca and Cb ($1 \leq Ca, Cb \leq 10^{12}$) separated by a space, they represent capacities of glass vases A and B respectively.

Output

A line containing an integer equal to the total amount of different water configurations in vases A and B if the only available operations are: completely fill a glass vase, completely empty a glass vase and pour water from one vase to another without water loss.

Examples

Input	Output
2 4	6

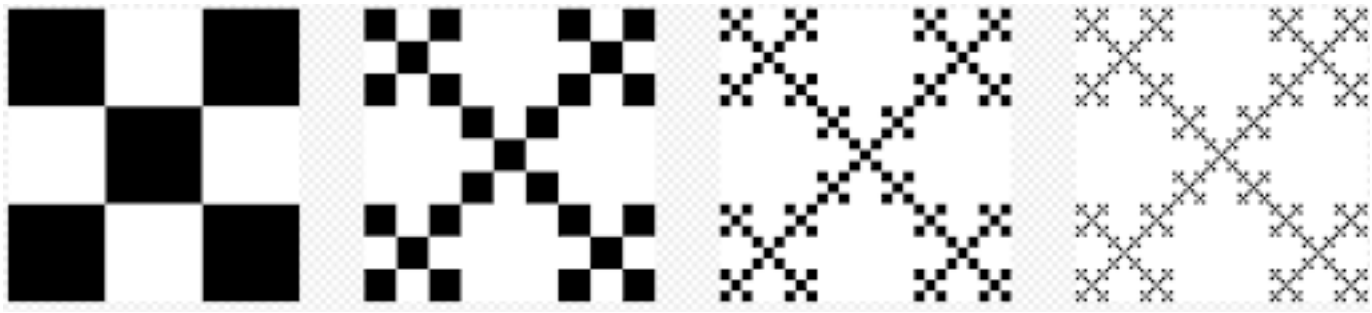
Example Output Explanation: All valid water configurations are:
$\{0,0\}, \{0,2\}, \{0,4\}, \{2,0\}, \{2,2\}, \{2,4\}$.

Image source Wikimedia https://upload.wikimedia.org/wikipedia/commons/thumb/5/58/Glass_ Vases_ and_ Bowl-BMA.jpg/1024px-Glass_ Vases_ and_ Bowl-BMA.jpg

Problem F. Vicsek's Fractal

Source file name:	Fractal.c, Fractal.cpp, Fractal.java, Fractal.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Gerson Yesid Lázaro Carrillo - UFPS (Graduate)`

Tamás Vicsek is a hungarian scientist recognized amongst other things, for having proposed the following fractal:



In this exercise we will draw the Vicsek's fractal using ASCII characters!

Starting from a base point, the fractal of order 0 will be a single square. To make the process easier, we will be using the character "#" to denote a square. Therefore the fractal of order 0 could be represented like this:

```
#
```

To draw the fractal of order 1, we place the fractal of orden 0 in the middle and we add a copy of it in each of its 4 corners like this:

```
# #
 #
# #
```

To draw the fractal of order 2, we place the fractal of order 1 in the middle and we draw a copy of it in each of its 4 corners:

```
# #   # #
 #     #
# #   # #
   # #
    #
   # #
# #   # #
 #     #
# #   # #
```

To summarize, in order to draw a fractal of order N (N > 0), we place the fractal of order n-1 in the middle and draw a copy of it in each of its 4 corners.

Input

Input contains a single number N ($0 \leq N \leq 7$), the order of the fractal to be drawn.

Output

The output for the exercise is the draw of the Vicsek's fractal of order N, using the character # to represent the squares and whitespaces in the other cells.

Examples

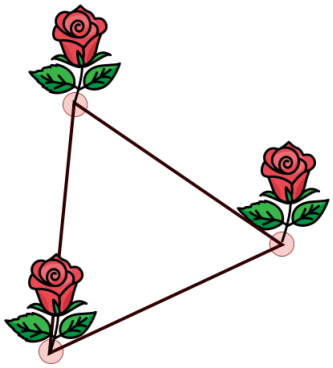
Input	Output
2	#.#...#.# .#.....#. #.#...#.# ...#.#... ....#.... ...#.#... #.#...#.# .#.....#. #.#...#.#

Note: It is important that all the empty spaces within the draw be whitespaces. In the example all the whitespaces have been replaced by dots (.) just for visualization purposes. Be sure that your output use whitespaces.

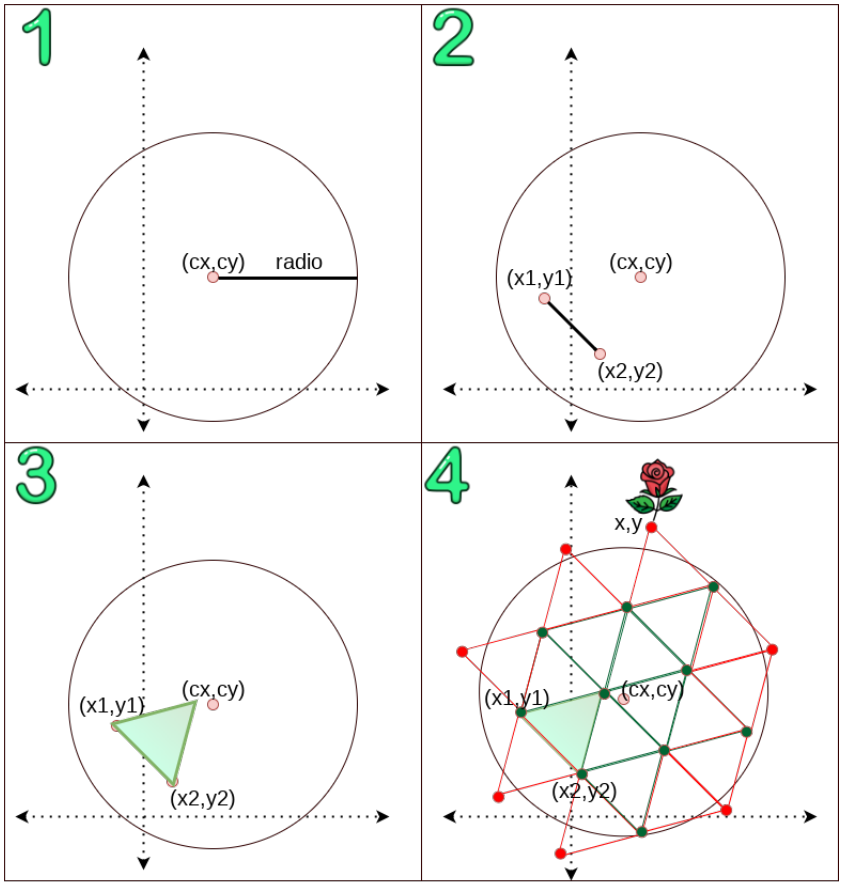
Problem G. Gardener

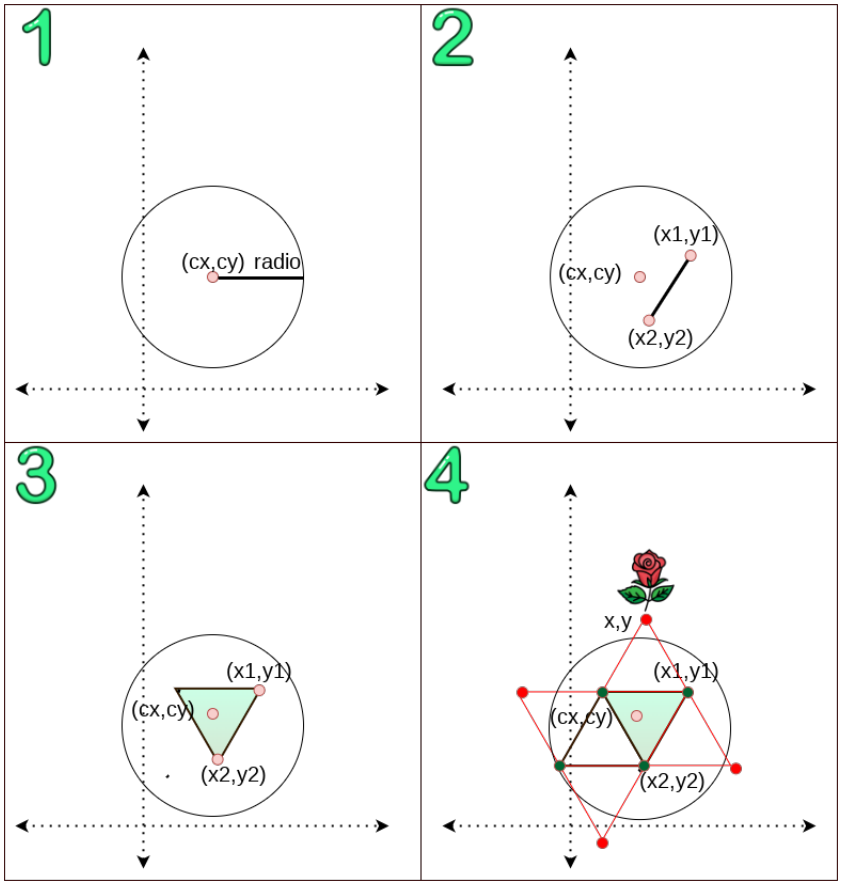
Source file name:	Gardener.c, Gardener.cpp, Gardener.java, Gardener.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Milton Jesús Vera Contreras - UFPS (Professor)`

Tresbolillo is a seeding system in which the plants form a triangle, usually isosceles or equilateral. Equilateral Tresbolillo is illustrated in the figure.



Two example scenarios are shown below and the Equilateral Tresbolillo process is described.





1. The area to sow is established: A circular area of radius $1 \leq r \leq 10^3$ and the location of the center, in Cartesian coordinates (cx, cy).
2. Two arbitrary points ($x1$, $y1$) and ($x2$, $y2$) are chosen within the area to sow, to start sowing. They determine the distance d that will be used for all equilateral triangles.
3. The first equilateral triangle is assembled with the two previous points ($x1$, $y1$) and ($x2$, $y2$) and the first three plants are sown.
4. Sowing continues in the form of an equilateral triangle, above, below and to the sides of each seeded triangle, without repeating triangles and until the largest extension of area to sow is covered. Do not continue sowing beside a triangle that have some point outside the circular area.

There are at least three drawbacks to this procedure:

- Some flowers are out of the area to sow.
- Sometimes there are more flowers outside the area than inside.
- Sometimes the area to sow is wasted, because the points to start sowing are arbitrary.

But these drawbacks can be ignored when it comes to Wilfrido Vargas's Gardener, forerunner of FreeStyle in the days of the teachers of your teachers ;) If you don't understand this last sentence, don't worry, it's an old song from the last century, that you can listen to while you solve the problem and dance to it when you solve it ;)

Wilfrido Vargas, Eddy Herrera, the teachers of your teachers and your teachers are very old to sow with Trestobillo and they want to program a robot to do it for them. May you help them?

Input

Several test cases, each test case on a different line. Each test case consists of seven (7) integers separated by whitespace: the coordinates of the center of the area to sow (cx, cy), the radius $1 \leq r \leq 10^3$ and the coordinates of the two points where to start sowing ($x1$, $y1$) and ($x2$, $y2$). The coordinates of the points ($x1$, $y1$) and ($x2$, $y2$) are always within the area to sow, they can be entered in any order and the segment they determine can have any inclination. The distance d between the points where you should start sowing ($x1$, $y1$) and ($x2$, $y2$) is at least one (1) and no more than the radius (r) of the area: $1 \leq d \leq r$.

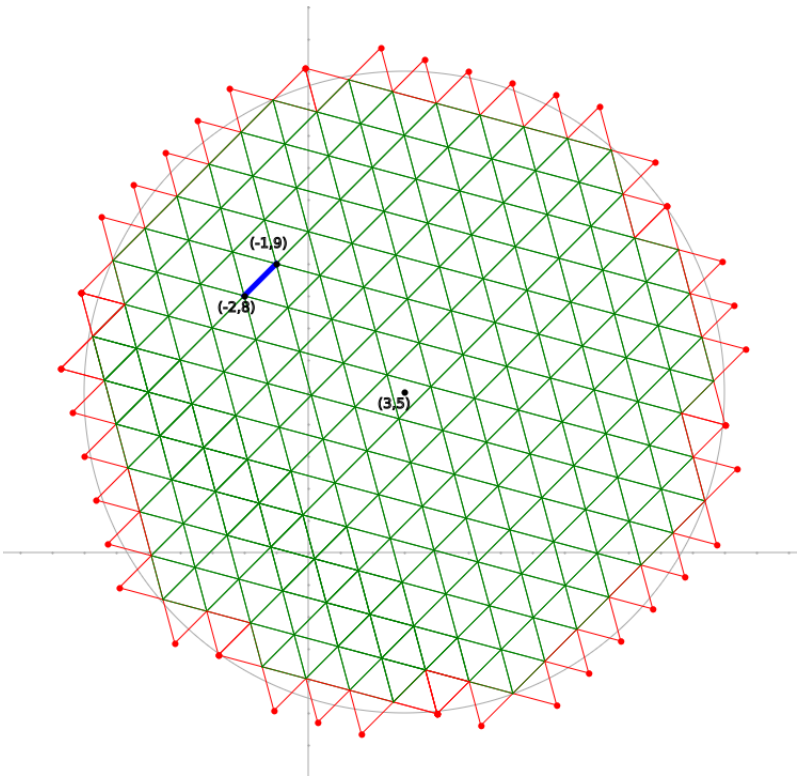
Output

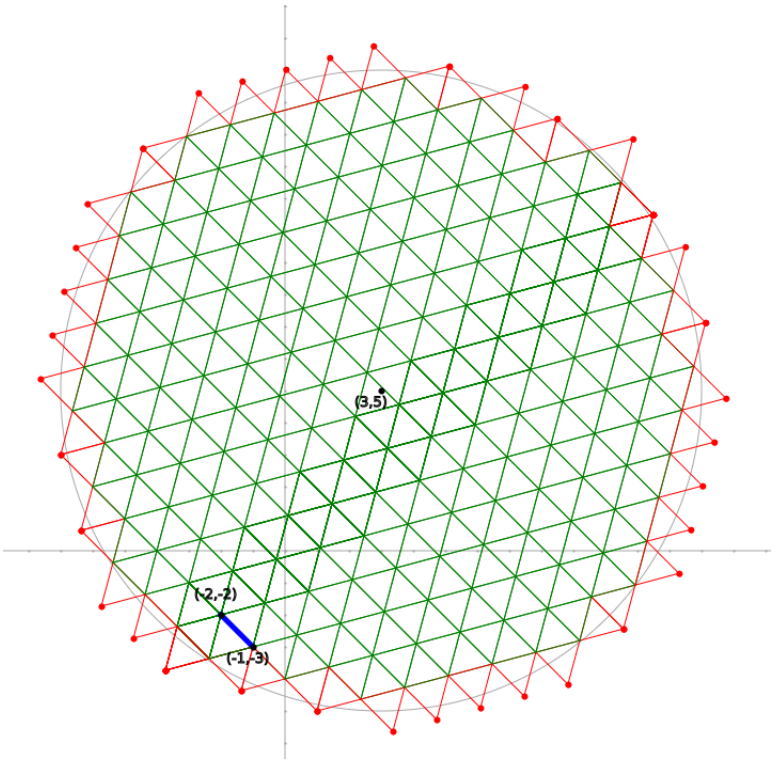
For each test case an integer M is printed that indicates the number of plants that were sown outside the field. If a point (x, y) is on the edge of the circular area, it is considered inside.

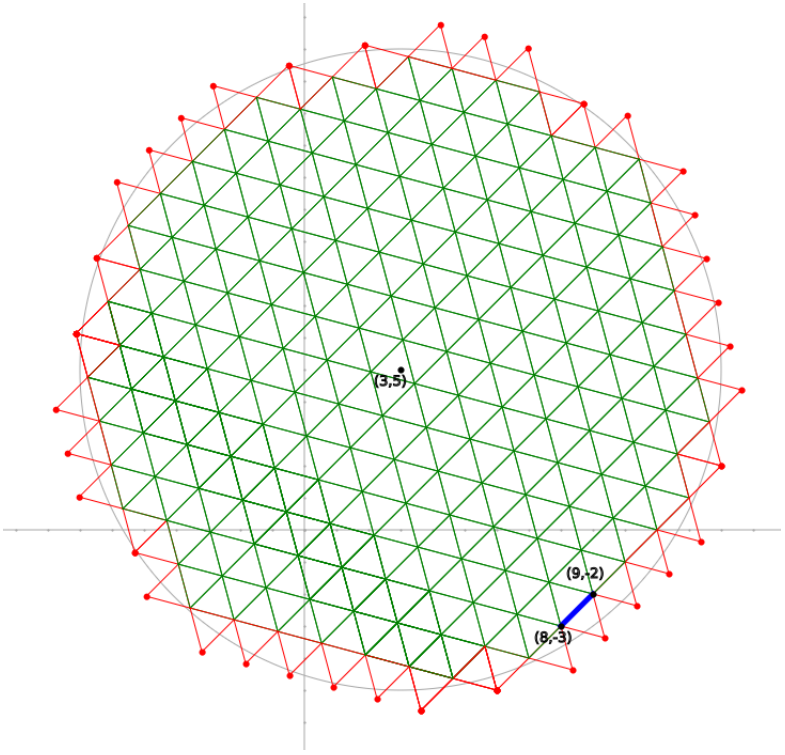
Examples

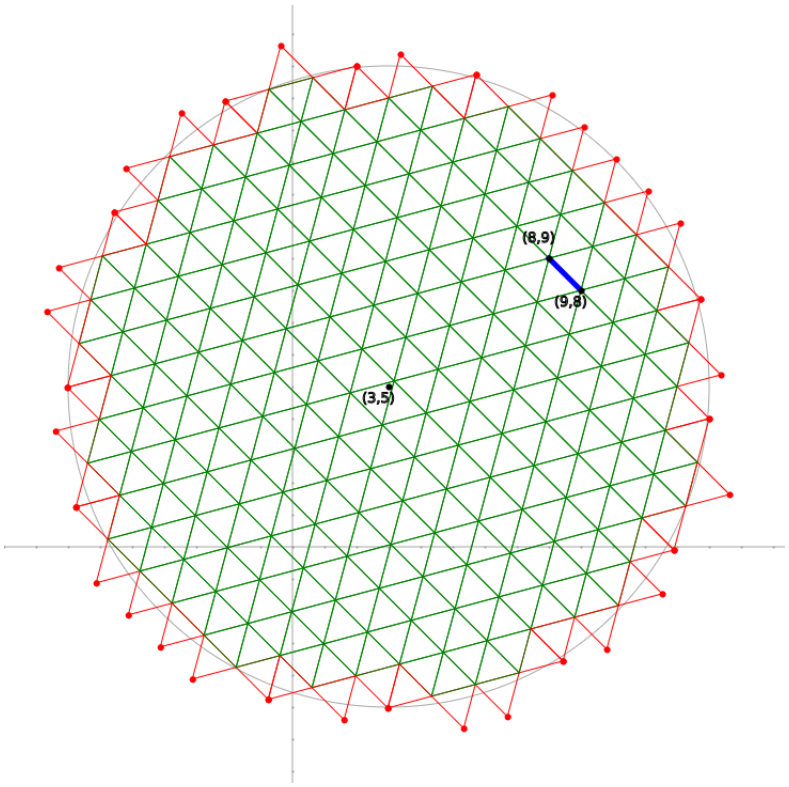
Input	Output
3 5 10 -2 8 -1 9	39
3 5 10 -2 -2 -1 -3	36
3 5 10 8 -3 9 -2	37
3 5 10 8 9 9 8	35
11 13 30 -9 3 9 27	4

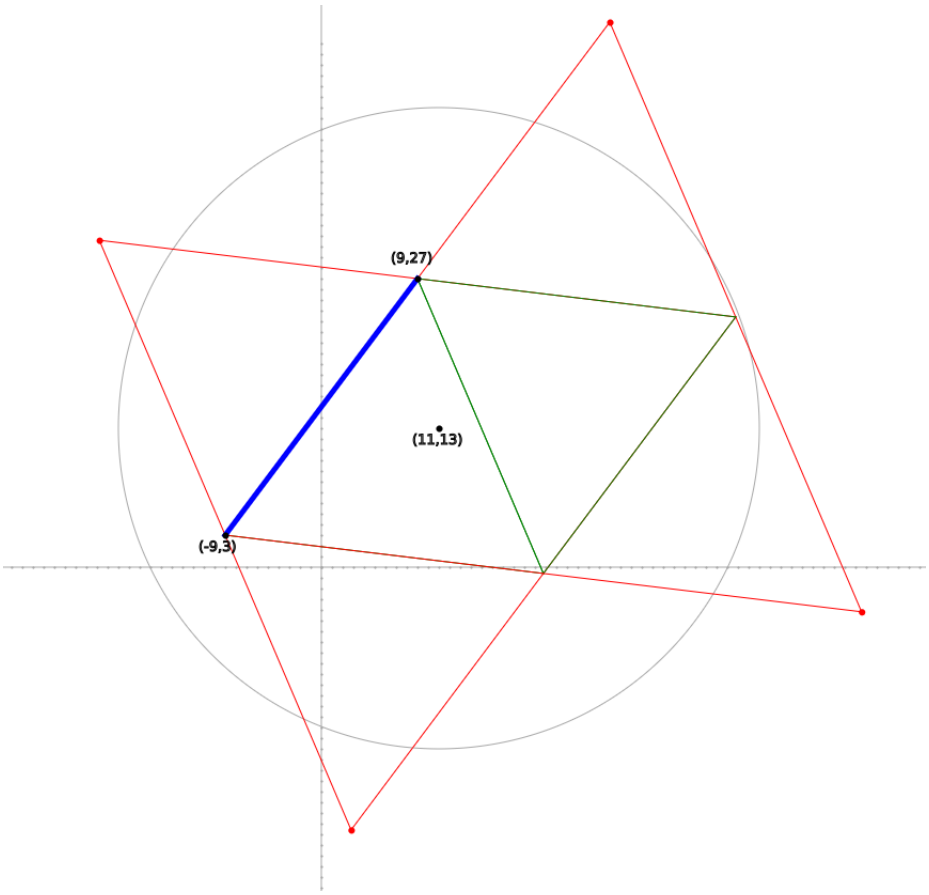
As precision error use $epsilon = 10^{-6}$
Trying to be exhaustive, in this challenge we will use 2^8 **test cases**
Below are the figures of the examples:




(3,5)
(-2,-2)
(-1,-3)


(3,5)
(9,-2)
(8,-3)


(8,9)
(9,8)
(3,5)


(9,27)
(11,13)
(-9,3)

Problem H. Harry Potter and the Portkey Problem

Source file name:	Harry.c, Harry.cpp, Harry.java, Harry.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Angie Melissa Delgado León - UFPS (Graduate)`

It's been hard times since the Ministry of Magic fell and **He-Who-Must-Not-Be-Named** took over. Harry, Ron and Hermione are safe for now in the ancestral home of the Black family, the Number 12 Grimmauld Place, but they need to continue with the mission that Dumbledore commissioned them. For this, they need to constantly move between n different places numbered from 1 to n, which include safe houses that the Order of the Phoenix has arranged for them, as well as places that have a strong connection with the Dark Lord's past.

Situation gets worse when they start to suspect that Harry continues to have the trace on him, which forbids them to use apparition as their transportation method. Due to all this, they have had to resort to Portkeys as their main travel mechanism. As we all know, Portkeys are magical objects enchanted to instantly bring anyone touching it to a specific location (including their current location). However, the limitation of the Portkeys is that they can be used only once.

Portkeys are currently quite coveted objects and regulated by the Ministry, so the golden trio has had to turn to the buyer and purveyor of wondrous objects, Mundungus-Fletcher. Mundungus can offer the group for a small price basic Portkeys from a location u to a location v, but since the trio does not trust the famous thief at all they can opt for 2 types of improved Portkeys too:

- Enhanced fixed origin Portkey: This Portkey can be used to travel from a fixed origin u to any destination in the range $[l, r]$ given by Mundungus.
- Enhanced fixed destination Portkey: This Portkey can be used to travel from any location in the range $[l, r]$ (given by Mundungus), to a fixed destination u.

The trio is not really sure about what their next move will be, but they want to be prepared for when they get a clue and have to move on. For this, they want to know the minimum amount of money they will have to spend on Portkeys to get from Grimmauld Place to any other destination (including Grimmauld Place).

Input

The first line of the input contains three integers n, p and g ($1 \leq n, p \leq 10^5; 1 \leq g \leq n$):

- n: number of locations the trio needs to travel to,
- p: number of Portkey options offered by Mundungus and
- g: index of Grimmauld Place.

The next p lines contain the Portkeys description. Each line starts with a string s, defining the type of Portkey ($basic$, $fixed-origin$, $fixed-destination$). If the Portkey is the type $basic$, then it is followed by three 3 integers v, u and w where w is the cost of that Portkey ($1 \leq v, u \leq n, 1 \leq w \leq 10^9$). Otherwise it is followed by four 4 integers u, l, r and w where w is the cost of that Portkey ($1 \leq u \leq n, 1 \leq l \leq r \leq n, 1 \leq w \leq 10^9$).

Output

Print a single line with n integers separated by spaces. The i-th of them should be the cost to get from Grimmauld Place to i-th the location, or -1 if it's impossible to get to that location.

Examples

Input	Output
6 8 1 fixed-origin 4 3 6 95 fixed-destination 3 1 3 24 fixed-destination 6 4 4 12 fixed-origin 5 1 2 9 basic 3 3 20 fixed-destination 4 2 6 32 basic 1 6 25 fixed-origin 6 1 5 6	0 31 24 31 31 25
5 4 1 fixed-origin 2 3 4 10 basic 2 4 16 fixed-destination 4 1 3 12 basic 2 5 25	0 -1 -1 12 -1

Image source https://www.tshirtroundup.com/tag/harry-potter/page/2

Problem I. Ivan Looks for a Console

Source file name:	Ivan.c, Ivan.cpp, Ivan.java, Ivan.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica (Professor)`

Iván Camilo Padilla Contreras (ICPC) wants to buy a new console. And he really likes variety, but has a limited budget. There are a large number of consoles on the market, each of which has a limited number of games. Ivan wants to know the maximum number of games he could have if he bought a single console that fits his budget.

Input

The input consists of several test cases. The first line contains a positive integer T ($1 \leq T \leq 100$), which is the number of test cases. Each test case begins with a line containing two positive integers separated by whitespace: C B ($1 \leq C \leq 10^4$, $1 \leq B \leq 10^6$), which respectively mean the number of consoles available on the market and the amount of money that Iván has as a budget for the purchase of the console. Then C lines are presented, each of them has positive integers separated by whitespace: C_i G_i ($1 \leq C_i \leq 10^5$, $1 \leq G_i \leq 10^4$), which indicate the cost and the number of games that the console has.

Output

For each test case, print in a single line the maximum amount of games that Ivan may play.

Examples

Input	Output
1 3 10 14 100 9 15 8 17	17

Image source Wikimedia https://upload.wikimedia.org/wikipedia/commons/3/39/Video-game-console-2202585_1920.jpg

Problem J. Join to ICPC's games if you like popularity

Source file name:	Join.c, Join.cpp, Join.java, Join.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica (Professor)`

Iván Camilo Padilla Contreras (ICPC) now has his console, but he has to choose again, he has a limited budget and he wants to maximize the popularity of his games. He has statistics on how many times a game has been purchased, therefore, how much popular it is. So, the number of times a game has been sold, shows how popular it is. Ivan wants to know how much "popularity" he can get with his budget. He measures the popularity by adding up all the sales of the games he got.

Input

The input is a single case. The first line has two integers, the first G ($1 \leq G \leq 10^4$) that denotates the number of games Ivan can buy, and a number B ($1 \leq B \leq 10^6$), that denotates his budget. The next G lines comes with two integers, P ($1 \leq P \leq 10^5$) and S ($1 \leq S \leq 10^8$), denotating the price and the number of sales respectively.

Output

A single number indicating the maximum popularity Ivan can get.

Examples

Input	Output
3 10 9 11 5 6 5 6	12

Image source Wikimedia https://upload.wikimedia.org/wikipedia/commons/3/39/Video-game-console-2202585_1920.jpg

Problem K. Kidnapped

Source file name:	Kidnapped.c, Kidnapped.cpp, Kidnapped.java, Kidnapped.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`3 second / 256 megabytes (details on page 7)`
Author(s):	`Wilmer-Emiro Castrillón-Calderón - U. de la Amazonía (Graduate)`

An emergency has occurred; your friend Manuel has been kidnapped; He is locked up in a labyrinth, he can move in four directions inside the labyrinth, up, down, left and right, represented by U, D, L and R, respectively. He needs to find the way out, but the labyrinth contains many mines, is very dangerous! Now you must rescue him.

Fortunately, your friend Diana found a map of the labyrinth and she discovered that the only safe route is the shortest path and lexicographically greater. Now you must write a program to find the safe route, ¿Can you help him?

Input

The input begin with $T \leq 100$, the number of test cases, the next line contains R and C ($2 \leq R, C \leq 500$) indicating the number of rows and columns of the labyrinth, the next R lines contains C characters describing the labyrinth, those lines contain the following characters: '`S`' the starting position of your friend, '`X`' the exit, '`#`' a wall (this can't be traversed), and '`.`' a free space (your friend can walk in these places). The characters '`S`' and '`X`' only appear once in the labyrinth.

Output

For each test case print a line with the safe path, if it doesn't exists then print *No exit*.

Examples

Input	Output
3 4 3 ##X ... ..# S## 4 3 #X# ... ### .S. 5 5 S.### #.... #.##. #.##. #...X	UURRU No exit RDRRRDDD

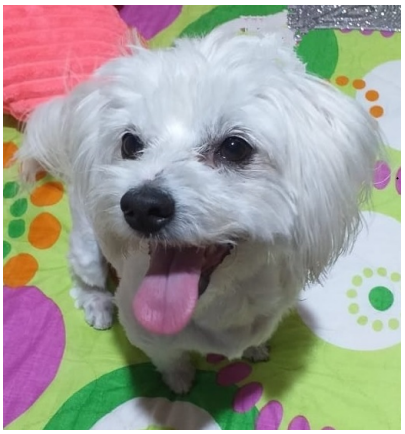
Problem L. Lucky is a Crazy Dog

Source file name:	Lucky.c, Lucky.cpp, Lucky.java, Lucky.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia (Professor)`

Lucky is a crazy dog. It runs and barks around the house all day.

The *Lucky's* house is like a maze that is layout as a grid (a matrix). Each cell in the matrix can be a wall (which cannot be traversed) or it can be a floor (which *Lucky* can run through). Lucky can move to a new cell from his current position if they are both adjacent and both are one floor.

Lucky wants to know how many floor cells he can run through and bark like crazy, taking a floor cell as a starting point. Obviously, *Lucky* is very busy all day fulfilling his duties, for this reason it needs your help to solve this challenge.



Lucky is a crazy dog!

Input

The input begins with a positive integer T ($1 \leq T \leq 10$), the number of test cases.

Each test case consists of two parts:

1. The description of *Lucky*'s house.
2. Lucky's queries about the cells from which he wants to start running.

The description of *Lucky*'s house begins with a line containing two positive integers W (Width) H (High), ($3 \leq W, H \leq 1000$), the dimensions of the house. The next H lines contains W characters describing each cell:

1. '`.`' - to indicate that it is a floor cell (*Lucky* can run through),
2. '`#`' - to indicate that it is a wall cell (which cannot be traversed).

Lucky's queries start with a positive integer Q $\left(1 \leq Q \leq \min\left(W \cdot H,\ 10^4\right)\right)$, the total queries. The next Q lines contains two positive integers R C ($1 \leq R \leq H$, $1 \leq C \leq W$), which represents the row and column of the cell from which *Lucky* wants to know how many cells he can reach (including the cell that is used as a starting point).

Output

For each test case, print a first line in the following format `Case idCase:`, where `idCase` is replaced by the test case number. Then print Q lines. On each line print the number of cells *Lucky* can reach from the starting point of that test case (including the cell that is used as a starting point). For more clarity on the input and output format look at the examples below.

Examples

Input	Output
2 6 5 ...#.. ..#... .#.### #.#... ...#.# 9 1 1 1 3 1 5 2 4 3 3 4 2 4 6 5 1 5 5 7 7 .#..#.. ..#.#.# ###.##. ...#... #.#.##. ..#...# .#..#.. 10 1 1 1 4 2 1 2 4 2 6 3 4 4 2 4 7 5 4 7 7	Case 1: 6 6 5 5 1 4 4 4 4 Case 2: 3 4 3 4 3 4 7 5 8 8

Problem M. Mixture

Source file name:	Mixture.c, Mixture.cpp, Mixture.java, Mixture.py
Input:	`standard input`
Output:	`standard output`
Time / Memory limit:	`1 second / 256 megabytes (details on page 7)`
Author(s):	`Juan Manuel Reyes - ICESI (Professor)`

A chemical compound is a substance formed by the combination of two or more elements of the periodic table. Mixture are represented by a chemical formula. For example: water(H_2O) consist of two hydrogen atoms and one oxygen atoms. The elements of a chemical compound cannot be divided or separated by physical processes (decantation, filtration, distillation), only by chemical processes. The smallest unit of a chemical compound (the one that cannot be physically separated) is called a molecule. Molecules are entities formed by atoms that have strong links between them and that make them sufficiently stable.

A mixture consists of two or more chemical compounds united, but not chemically combined. In a mixture, a chemical reaction does not occur and each of its components maintains its identity and chemical properties. Some mixtures can be separated into their components by physical processes (mechanical or thermal), such as distillation, dissolution, magnetic separation, flotation, sieving, filtration, decantation or centrifugation.

A mixture can be homogeneous or heterogeneous. Homogeneous mixtures are those in which the intensive properties are the same throughout the mixture; it is uniform in appearance and has the same properties as a whole (for example, salt dissolved in water). In homogeneous mixtures their compounds cannot be differentiated with the naked eye. Heterogeneous mixtures are those in which the parts maintain different intensive properties (eg sand mixed with sawdust). Heterogeneous mixtures have a non-uniform composition in which its components can be distinguished with the naked eye, it is formed by two or more physically different substances, unevenly distributed.

The teacher Adenalleva Oibaf has made observations of interesting substances, he has obtained information on the atoms that compose them and the bonds that join those atoms with others in the same molecule. Since a substance can be a pure compound (that is, made up of a single compound) or a mixture (that is, made up of more than one compound), the teacher is interested in knowing how many chemical components each of the compounds are made of observed substances.

Input

The first line is a number $n < 100$, with the total of substances. Then the description of n substances.

Each substance has several lines: the first line has two positive integers $1 < a < 1000$ y $0 < e < 10000$, separated by a space. The a number is the number of atoms found in the substance y the e number is the number of links between those atoms. The next e lines contains two positive integers x y, separated by a space, indicating that this pair of atoms have a link to each other. In other words, the atom x has a link with the atom y.

Output

For each substance print n lines, one for each substance, in which it says: "`Pure Compound!`" if there is only one chemical compound in the substance, and "`Mixture:` m", if the substance has more than one compound, where m is the number of compounds in that substance.

Examples

Input	Output
2 5 4 0 2 2 4 1 3 0 4 4 4 0 1 2 0 1 3 2 1	Mixture: 2 Pure Compound!

Image source Wikimedia https://commons.wikimedia.org/wiki/File:Chemistry-3533039_960_720.jpg