

Maratón de Programación 2021



2015



2016



2017



2017



2018



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

Grupo de Estudio

Programación Competitiva



12:00 Medio día

hasta las 7:00 p.m.



SÁBADO

11 de Diciembre

de 2021

08

FECHE

HORA

VIRTUAL

LUGAR

2019



Maratón UFPS 2020

Contents

1	Séptima Maratón de Programación UFPS - 2021	2
2	Grupo de Estudio en Programación Competitiva	4
3	Instrucciones	4
4	Reglas	4
5	Lista de Problemas	6
6	Fe de Erratas	8

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)

Este trabajo se comparte bajo licencia Creative Commons Reconocimiento-Compartir Igual 4.0 Internacional (CC BY-SA 4.0)





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 más problemas. Entre dos equipos que resuelven 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 ;)

1. A Twin Prime Number (Page 9)
2. Beginner: this is your challenge (Pages 10-12)
3. Carlos's Game (Pages 13-14)
4. Dangie's Function (Page 15-16)
5. Enjoys playing with glass vases (Page 17)
6. Vicsek's Fractal (Pages 18-19)
7. Gardener (Pages 20-24)
8. Harry Potter and the Portkey Problem (Pages 25-26)
9. Ivan Looks for a Console (Page 27)
10. Join to ICPC's games if you like popularity (Page 28)
11. Kidnapped (Page 29)
12. Lucky is a Crazy Dog (Pages 30-31)
13. Mixture (Pages 32-33)

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						
A	A Twin Prime Number	1	3	2	5	5	2	2	5	5
B	Beginner: this is your challenge	1	3	2	5	5	2	2	5	5
C	Carlos's Game	1	6	1	45	1	1	1	7	7
D	Dangie's Function	1	1	2	5	5	2	2	5	5
E	Enjoys playing with glass vases	1	2	1	7	1	1	1	7	7
F	Vicsek's Fractal	1	2	1	2	1	1	1	2	2
G	Gardener	1	60	5	90	5	5	5	90	90
H	Harry Potter and the Portkey Problem	1	3	6	30	6	6	6	30	30
I	Ivan Looks for a Console	1	1	2	5	5	2	2	5	5
J	Join to ICPC's games if you like popularity	1	1	2	5	5	2	2	5	5
K	Kidnapped	1	1	2	2	2	2	2	2	2
L	Lucky is a Crazy Dog	1	1	2	5	5	2	2	5	5
M	Mixture	1	1	5	5	5	5	5	5	5

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

6 Fe de Erratas

Errare humanum est, lo importante es reconocer y corregir el error. Durante esta competencia se presentaron los siguientes errores:

1. Por seguridad se cargó un cuadernillo temporal que no correspondía a la competencia. Los maratonistas fueron mucho más rápidos que el equipo a cargo de la maratón y alcanzaron a realizar envíos con el cuadernillo equivocado. El ajuste se realizó unos dos minutos después de la competencia y se envió la clarificación. Los casos juzgados bajo este error fueron eliminados y no afectó a los competidores.
2. El problema F fue configurado con un tamaño de fichero de salida por defecto de 2048 kbytes. Y desde el caso 6 se requería un tamaño mayor. Se hizo el ajuste, se envió la clarificación y se rejujgaron los casos. Los casos juzgados bajo este no afectó a los competidores.
3. El Problema J tuvo un cambio en los casos de prueba que no fue ajustado en el cuadernillo: $(1 \leq G \leq 10^4)$ cambió por $(1 \leq G \leq 10^3)$ y $1 \leq B \leq 10^6$ cambió por $1 \leq B \leq 10^4$.
4. El cuadernillo de RPC se elaboró en inglés y español. En esta versión, ajustada después de la competencia, agregamos, al final, la versión en español.
5. En el encabezado de algunas páginas aparecía año 2020 (la pandemia nos tiene congelados en el tiempo ;)... Se ajustó también este error.

Entendemos la inconformidad que estos errores generan, pero también confiamos en su comprensión. No es una tarea sencilla llevar a cabo una maratón de este tipo. Se requiere la colaboración voluntaria de muchas personas, quienes se esfuerzan mucho más allá de sus obligaciones. Y es algo que hemos logrado mantener durante siete años entre la UFPS y RPC, siempre con mucho entusiasmo y cariño para mantener viva la llama de la Programación Competitiva.

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)



Al profesor Elio Sabio le gustan los números primos gemelos. Un número primo gemelo es un número primo que es mayor o menor en dos unidades respecto a otro número primo.

El día del examen final del curso, el profesor decidió dar diferentes problemas a sus estudiantes, acerca de números primos y a ti te correspondió el siguiente:

Dado un número n , encontrar el número primo gemelo más cercano a n . Si hay dos números que tengan la misma distancia a n debes seleccionar el más grande.

Input

La primera línea contiene un número entero positivo $1 \leq t \leq 10^5$, que corresponde al número de casos de prueba. Luego hay t líneas, cada una con un número entero positivo $1 \leq n \leq 10^6$.

Output

Por cada caso de prueba debes generar una línea que contenga el número primo gemelo más cercano a n .

Examples

Input	Output
4	3
1	3
2	5
5	13
12	

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)

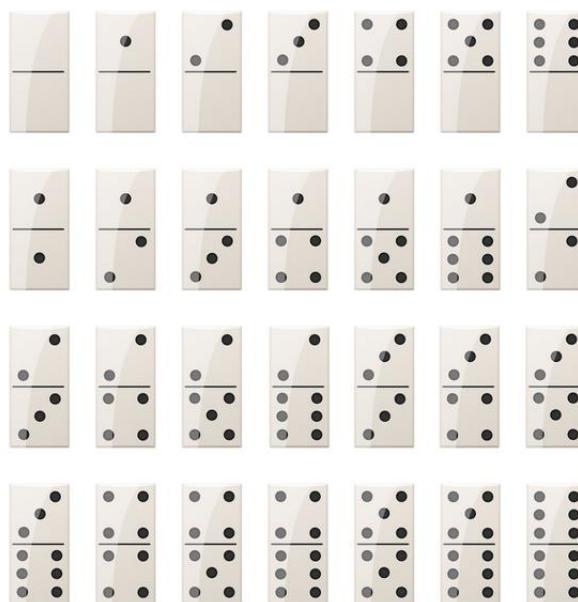
Author(s): Milton Jesús Vera Contreras - UFPS (Professor)

Este reto tiene el propósito de que el premio al mejor equipo novato lo ganen estudiantes de primeros semestres. Para que se motiven a continuar fortaleciendo la programación competitiva en la UFPS.

El juego de Dominó es antiguo, popular, sencillo y ¡misterioso!. Hay mucha matemática interesante en este juego...

Las reglas del juego se pueden resumir a lo siguiente:

- Es un juego basado en veintiocho (28) fichas, cada una de las cuales tiene dos extremos y en cada extremo hay p puntos ($0 \leq p \leq 6$), sin repeticiones, como en la figura:



- Las fichas se pueden distribuir entre dos (2), cuatro (4) o más jugadores, normalmente el número máximo de jugadores es cuatro (4).
- Cada ficha se puede representar por uno o dos números, dependiendo del orden de sus extremos. Por ejemplo, la ficha 15 es la misma ficha 51, la ficha 10 es la misma ficha 1 (el cero a la izquierda se ignora) y la ficha 34 es la misma ficha 43. Las fichas con dobles solo tienen un número que las representa: 0, 11, 22, 33, 44, 55 y 66.
- El juego se inicia siempre con una ficha doble, normalmente el doble seis 66, pero se puede acordar cualquier otro, como el doble cinco 55 o el doble cero 00.
- Las fichas deben ubicarse en una mesa, de manera adyacente, haciendo coincidir los dos extremos que tengan el mismo número de puntos p . Por ejemplo, con dos jugadores, se puede iniciar con 66, luego el 65, después el 52 y así sucesivamente. Esta regla de adyacencia es obligatoria para considerar que el juego es válido.

- Se sigue el juego hasta finalizar las fichas o hasta cerrar el juego, lo cual se explicará posteriormente.

Cualquier disposición de fichas, de cualquier cantidad k se puede representar mediante un número n , usualmente gigante, con la siguiente ecuación, que llamaremos la **Ecuación General del Dominó**:

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

Por ejemplo, la siguiente disposición de diez ($k = 10$) fichas corresponde al número 54.422.555.511.665.500.333 cincuenta y cuatro trillones cuatrocientos veintidos mil quinientos cincuenta y cinco billones quinientos once mil seiscientos sesenta y cinco millones quinientos mil trescientos treinta y tres (en colombiano):

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

Un caso particular del dominó es el **Juego Perfectamente Cerrado**. Este caso consiste en que los dos extremos del juego tienen la misma cantidad de puntos p y todas las siete (7) fichas que tienen esa cantidad de puntos en sus extremos ya están en la mesa, ya se han jugado. Por ejemplo, estos dos escenarios corresponden a juegos perfectamente cerrados con $p = 6$.

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

Un Juego Perfectamente Cerrado tiene un tamaño k , que corresponde a la cantidad de fichas que se han jugado. Nos interesan los casos de $k = 10$ que son los más pequeños y sencillos y de los cuales el profesor ha calculado más de diez mil 10^4 casos para este reto, todos similares a los dos anteriores. Para estos escenarios de diez fichas:

- Se puede identificar a simple vista si el juego es perfectamente cerrado.
- Si ese juego se representa mediante un número n con la ecuación descrita previamente, es sencillo escribir un programa de computador que determine si el juego es perfectamente cerrado.
- Si a ese número n se le desordenan algunos de sus dígitos, es decir, se aplica una permutación del número n , se obtiene un segundo número m , que no siempre representa un juego válido, no siempre cumple la ecuación general del dominó. Los dos escenarios de ejemplo anteriores son mutuamente una permutación y representan juegos válidos, cumplen la ecuación. Y la siguiente figura muestra dos permutaciones que no son juegos válidos, no cumplen la ecuación.

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

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

¿Quieres ganar el trofeo y las medallas del mejor equipo novato?

¡Resuelve este reto, puedes lograrlo!

Input

Cada caso de prueba consiste en un número m de veinte (20) dígitos d ($0 \leq d \leq 6$).

Output

Por cada caso de prueba imprimir en una línea tres avisos de SI (YES) o NO (NO), separados por un espacio en blanco, según se cumplan o no cada una de las siguientes condiciones:

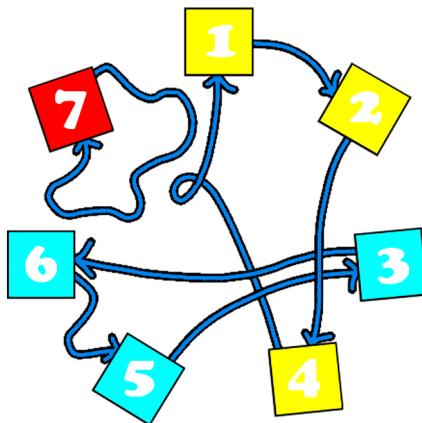
- El número m cumple la **Ecuación General del Dominó** y por lo tanto corresponde a un juego de dominó válido.
- El número m es una permutación de un número n y el número n representa un **Juego Perfectamente Cerrado** de tamaño diez $k = 10$.
- El número m representa un **Juego Perfectamente Cerrado** de tamaño diez $k = 10$.

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 es uno de los muchos estudiantes para quienes el cambio de clases presenciales a clases virtuales (de manera improvisada) convirtió sus días de estudiante universitario en algo no tan divertido como solía ser. Para pasar el tiempo Carlos, quien es muy inteligente y creativo, inventó un juego llamado “El Juego de Carlos” y consiste en lo siguiente:

Primero, Carlos elige un número entero positivo n y dibuja un tablero con n casillas numeradas de 1 a n organizadas en forma circular como un reloj (la casilla 2 está delante de la 1, la casilla 3 está delante de la 2, la casilla 1 está delante de la n , etc). Luego, dibuja caminos unidireccionales desde cada casilla i hacia la casilla que está i posiciones adelante en el tablero.

Una vez que el tablero está listo, Carlos elige dos casillas aleatorias x y y del tablero y coloca una ficha en cada casilla. Por último, elige dos números enteros positivos a y b que indican cuantas casillas se mueven las fichas en cada turno (siguiendo los caminos unidireccionales), respectivamente.

El reto de “El Juego de Carlos” es averiguar cuántos turnos deben pasar para que las dos fichas se encuentren por primera vez en una misma casilla o demostrar que las dos fichas nunca se encontrarán.

Carlos ha estado jugando por mucho tiempo y ya es un experto. Sabe que si $x = n$ o $y = n$, es muy fácil resolver el reto, por lo que ahora descarta esos casos. También piensa que no es divertido elegir valores pequeños para n de una manera tan general, por lo que ahora solo juega casos donde $n \geq 3$ y es un **número primo**. Además, n puede ser muy grande.

Actualmente jugar “El Juego de Carlos” es realmente difícil, tan difícil que Carlos ni siquiera ha vuelto a sus clases virtuales y lleva días tratando de resolver el reto. ¿Podrás resolverlo antes que él?

Input

La entrada consiste de una sola línea que contiene cinco enteros n , x , a , y y b ($3 \leq n \leq 10^{10}$ y **es primo**, $1 \leq x, y < n$, $1 \leq a, b \leq 10^{10}$) que indican los enteros elegidos por Carlos descritos anteriormente.

Output

La salida consiste de una sola línea. Si las dos fichas nunca se encuentran imprime -1 . De lo contrario, imprime dos enteros k y z indicando cuántos turnos deben pasar para que las dos fichas se encuentren por primera vez en la misma casilla y cuál será dicha casilla respectivamente.



Se garantiza que la respuesta cabe en un entero de 64 bits.

Examples

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

Note

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

Normalmente en un curso de programación de primer año en un programa académico de Ingenierías o de Ciencias de la Computación se le enseña a los estudiantes a construir funciones que hacen uso de los ciclos de repetición anidados como la siguiente:

```
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;
}
```

La complejidad temporal del algoritmo previo pertenece a $O(n^3)$, lo que implica que con el valor de $n = 10^6$, el número total de operaciones realizadas por el algoritmo, para poder dar el resultado requeriría 10^{18} pasos. Como un competidor de las diferentes fases de la ICPC, usted sabe que esa solución obtendrá el veredicto de **Time Limit Exceeded** en la competencia. Por esta razón se le pide proponer una solución que tenga una complejidad temporal tan cercana a $O(1)$ como sea posible, independientemente del tamaño de n . ¡Para ello requerirá de toda su experiencia como un competidor de la ICPC!

Input

La entrada comienza con un número entero positivo t ($1 \leq t \leq 10^6$), el cual representa el total de casos de prueba. Luego son presentadas t líneas cada una de ellas conteniendo un número entero positivo n ($1 \leq n \leq 10^6$) para el cual se debe calcular el resultado de la **Función de Dangie**.

Output

La salida debe contener t líneas, cada una de ellas conteniendo un entero positivo largo como resultado de la **Función de Dangie**.



Examples

Input	Output
7	0
1	40
5	5200
25	46852
52	192669568
833	47794715890
5234	33333333333000000
1000000	

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, el hechicero, se divierte jugando con jarrones de cristal. En esta ocasión cuenta con dos jarrones, A y B , con capacidades de Ca y Cb litros de agua respectivamente. Inicialmente ambos jarrones están vacíos. Magus puede llenar completamente cualquier jarrón con agua, vaciar por completo cualquiera de ellos o incluso verter agua de un jarrón a otro evitando la pérdida de agua. Ha estado llenando, vaciando y vertiendo agua en los dos jarrones durante al menos dos horas, por lo que perdió la cuenta de cuántas configuraciones diferentes de agua en los jarrones A y B había visto hasta ahora. Se requiere encontrar la cantidad total de diferentes configuraciones de agua en los jarrones A y B si las únicas operaciones disponibles son: llenar completamente un jarrón, vaciar completamente un jarrón y verter agua de un jarrón a otro sin pérdida de agua .

Input

La primera y única línea de entrada contiene dos números enteros positivos Ca y Cb ($1 \leq Ca, Cb \leq 10^{12}$) separados por un espacio, que representan las capacidades de jarrones A y B respectivamente.

Output

Una línea que contiene un número entero igual a la cantidad total de diferentes configuraciones de agua en jarrones A y B si las únicas operaciones disponibles son: llenar completamente un jarrón de vidrio, vaciar completamente un jarrón de vidrio y verter agua de un jarrón a otro sin perdida de agua.

Examples

Input	Output
2 4	6

Explicación del ejemplo: Todas las configuraciones de agua válidas son:

$\{0, 0\}, \{0, 2\}, \{0, 4\}, \{2, 0\}, \{2, 2\}, \{2, 4\}$.

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 es un científico hungaro reconocido entre otra cosas, por haber propuesto el siguiente fractal que lleva su nombre:



¡En este ejercicio dibujaremos el fractal de Vicsek con caracteres ASCII!

Partamos de un punto base, el fractal de orden 0 que sería simplemente un cuadrado. Para hacerlo más sencillo, usaremos el carácter “#” como cuadrado. Por tanto el fractal de orden cero sería:

#

Para dibujar el fractal de orden 1, ubicamos el fractal de orden 0 en la mitad, y una copia de él en cada una de sus 4 diagonales así:

#

Para dibujar el fractal de orden 2, ubicamos el fractal de orden 1 en el centro, y dibujamos una copia de él en cada una de sus 4 diagonales:

#

Generalizando, para dibujar un fractal de orden N ($N > 0$), ubicamos el fractal de orden $(N - 1)$ en el centro, y dibujamos una copia de él en cada una de sus 4 diagonales:

Input

La entrada contiene un número N ($0 \leq N \leq 7$), el orden del fractal a dibujar.

Output

La salida es el dibujo del fractal de Vicsek de nivel N , usando el carácter “#” para representar los cuadrados y espacios en blanco en las demás casillas.

Examples

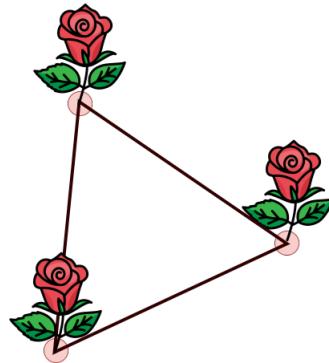
Input	Output
2	#.#...#.# .#.....#. #.#...#.# ...#.#...#.... ...#.#... #.#...#.# .#.....#. #.#...#.#

Note: Es importante que todos los espacios vacíos dentro de la representación sean espacios en blanco. En el ejemplo se han reemplazado todos los espacios en blanco de la salida por puntos (.) solo con fines de una mejor visualización, asegúrese que su salida use espacios.

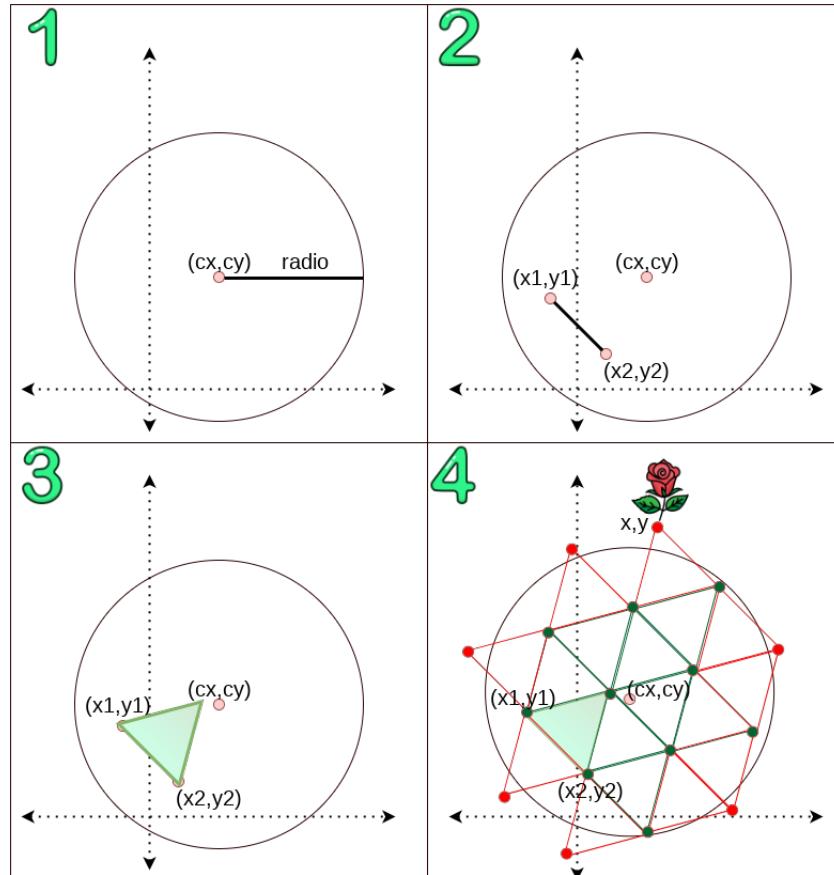
Problem G. Gardener

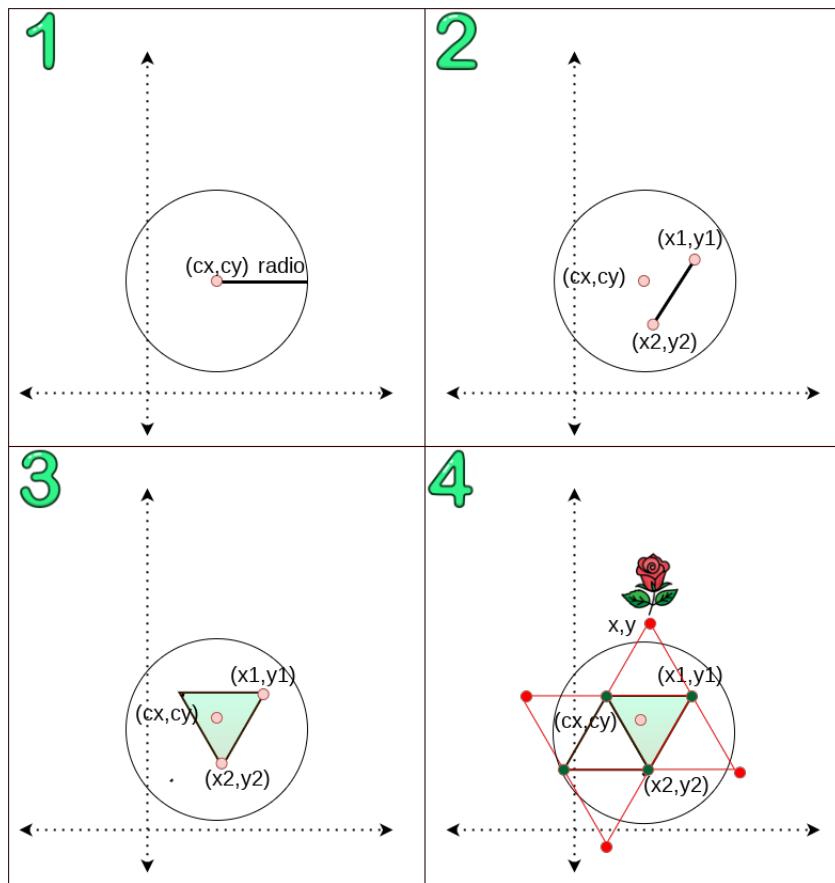
Source file name: Gardener.c, Gardener.cpp, Gardener.java, Gardener.py
 Input: standard input
 Output: standard output
 Time / Memory limit: 1 second / 256 megabytes (details on page 7)
 Author(s): Milton Jesús Vera Contreras - UFPS (Professor)

Tresbolillo es un sistema de sembrado en el que las plantas forman un triángulo, normalmente isósceles o equilátero. En la figura se ilustra Tresbolillo equilátero.



A continuación se muestran dos escenarios de ejemplo y se describe el proceso de Trestobillo equilátero.





1. Se establece el terreno donde se va a sembrar: Un área circular de radio $1 \leq r \leq 10^3$ y la ubicación del centro, en coordenadas cartesianas (cx, cy) .
2. Se escogen dos puntos arbitrarios (x_1, y_1) y (x_2, y_2) , dentro del terreno, para empezar a sembrar, los cuales determinan la distancia d que se usará para todos los triángulos equiláteros.
3. Se arma el primer triángulo equilátero con los dos puntos anteriores (x_1, y_1) y (x_2, y_2) y se siembran las tres primeras plantas.
4. Se sigue sembrando en forma de triángulo equilátero, por encima, debajo y a los lados de cada triángulo sembrado, sin repetir triángulos y hasta cubrir la mayor extensión del terreno. No se sigue sembrando junto a los triángulos que tengan algún punto por fuera del terreno circular.

Como se puede apreciar en las figuras, hay, por lo menos, tres inconvenientes de este procedimiento:

- Algunas flores quedan por fuera del terreno.
- En algunas ocasiones hay más flores fuera del terreno que adentro.
- En algunas ocasiones se desperdicia el terreno, debido a que los puntos para empezar a sembrar son arbitrarios.

Pero estos inconvenientes se pueden ignorar cuando se trata de El Jardinero de Wilfrido Vargas, precursor del FreeStyle en los tiempos de los profesores de sus profesores ;) Si no entiende esta última oración, no se preocupe, es una vieja canción del siglo pasado que puede oír mientras resuelve el problema y bailarla cuando lo resuelva ;)

Wilfrido Vargas, Eddy Herrera, los profesores de sus profesores y sus profesores ya no están en edad de sembrar con Trestobillo y quieren programar un robot que se encargue de hacerlo por ellos. ¿Puedes ayudarles?

Input

Varios casos de prueba, cada uno en una línea diferente. Cada caso de prueba consiste en siete (7) números enteros separados por espacio en blanco: las coordenadas del centro del terreno (cx, cy), el radio del terreno $1 \leq r \leq 10^3$ y las coordenadas de los dos puntos donde se debe empezar a sembrar ($x1, y1$) y ($x2, y2$). Las coordenadas de los puntos ($x1, y1$) y ($x2, y2$) siempre están dentro del terreno, pueden ingresar en cualquier orden y el segmento que determinan puede tener cualquier inclinación. La distancia d entre los puntos donde se debe empezar a sembrar ($x1, y1$) y ($x2, y2$) es mínimo uno (1) y máximo el radio (r) del terreno: $1 \leq d \leq r$.

Output

Por cada caso de prueba se imprime un entero M que indica el número de plantas sembradas por fuera del terreno. Si un punto (x, y) está en el borde del terreno se considera por dentro.

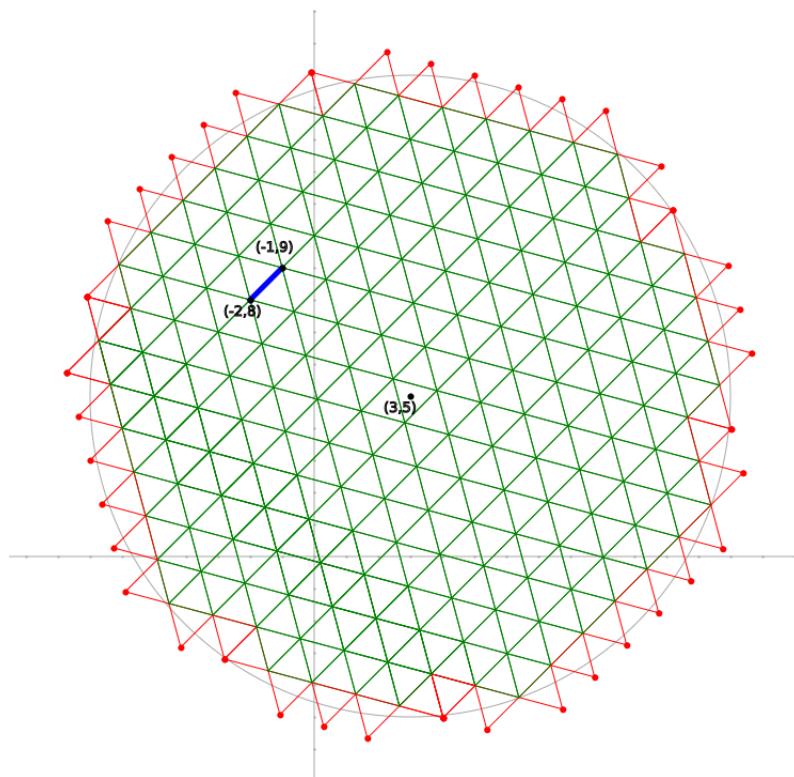
Examples

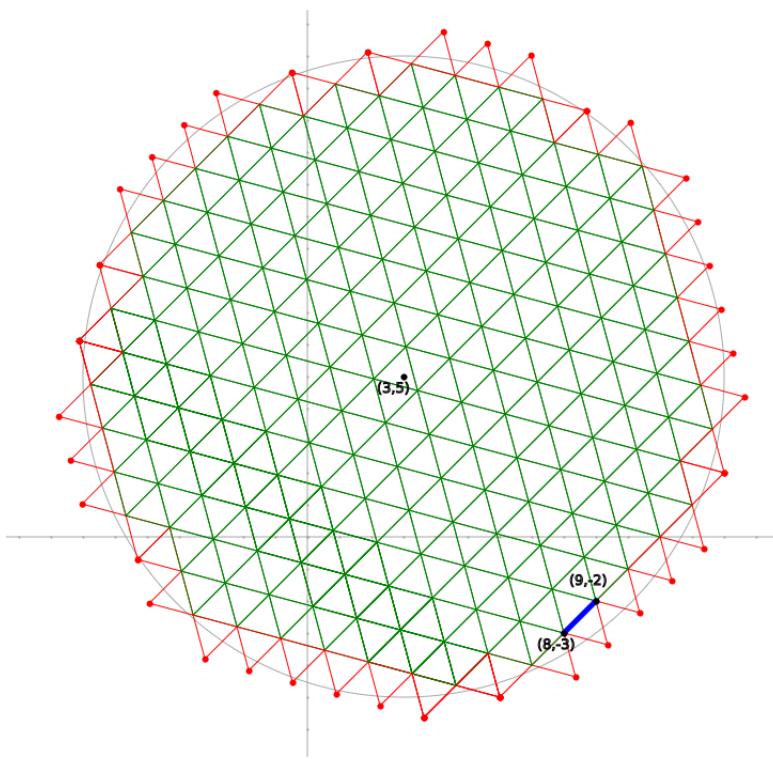
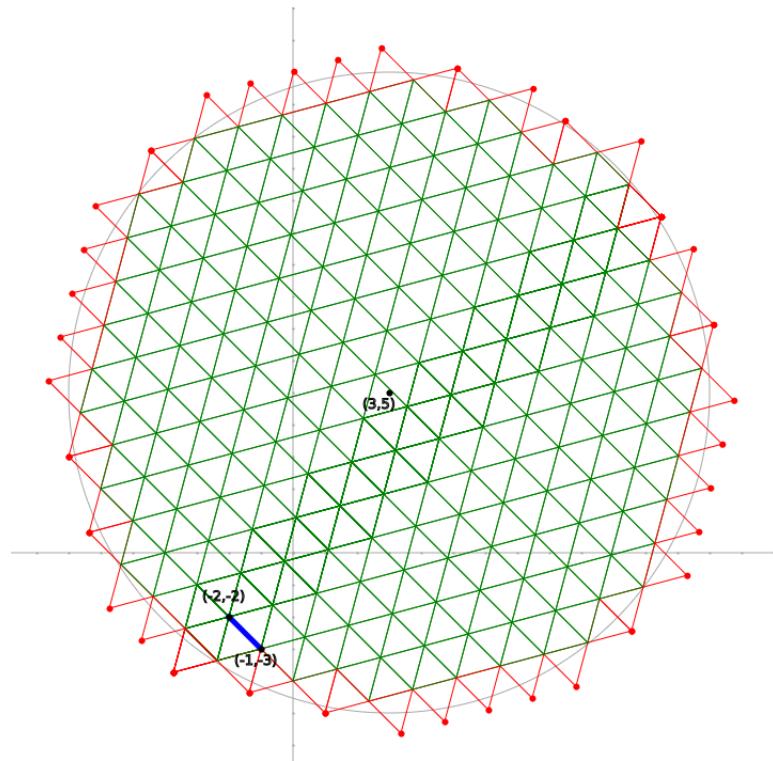
Input	Output
3 5 10 -2 8 -1 9	39
3 5 10 -2 -2 -1 -3	36
3 5 10 8 -3 9 -2	37
3 5 10 8 9 9 8	35
11 13 30 -9 3 9 27	4

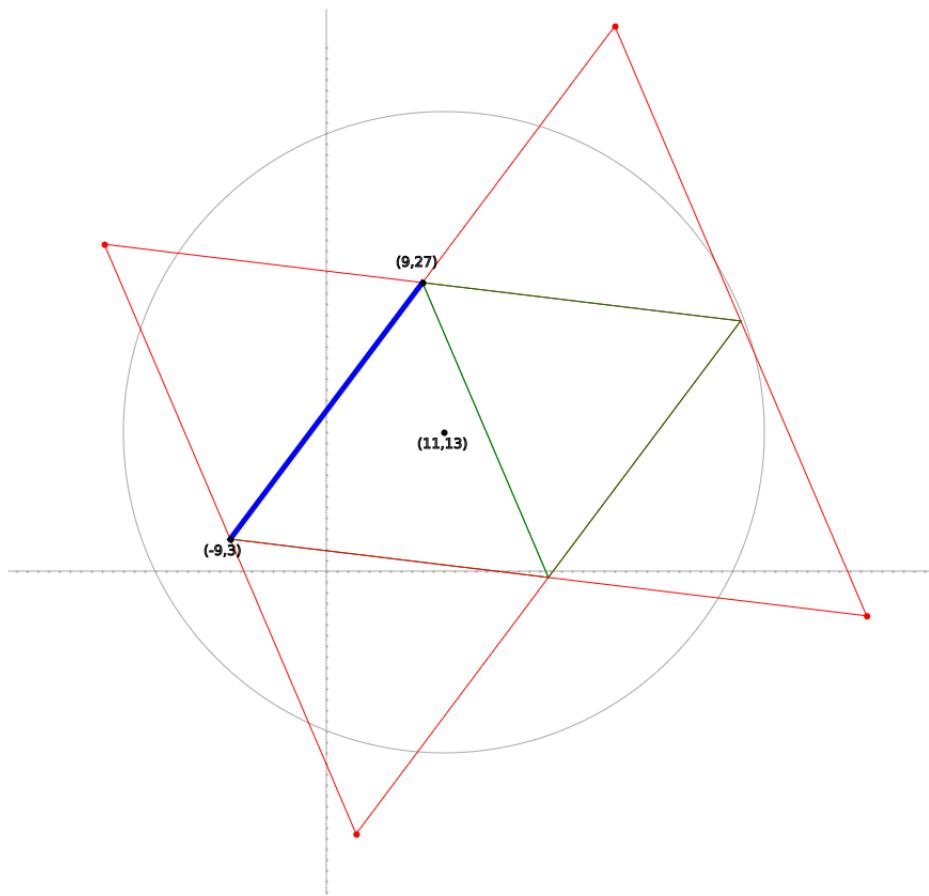
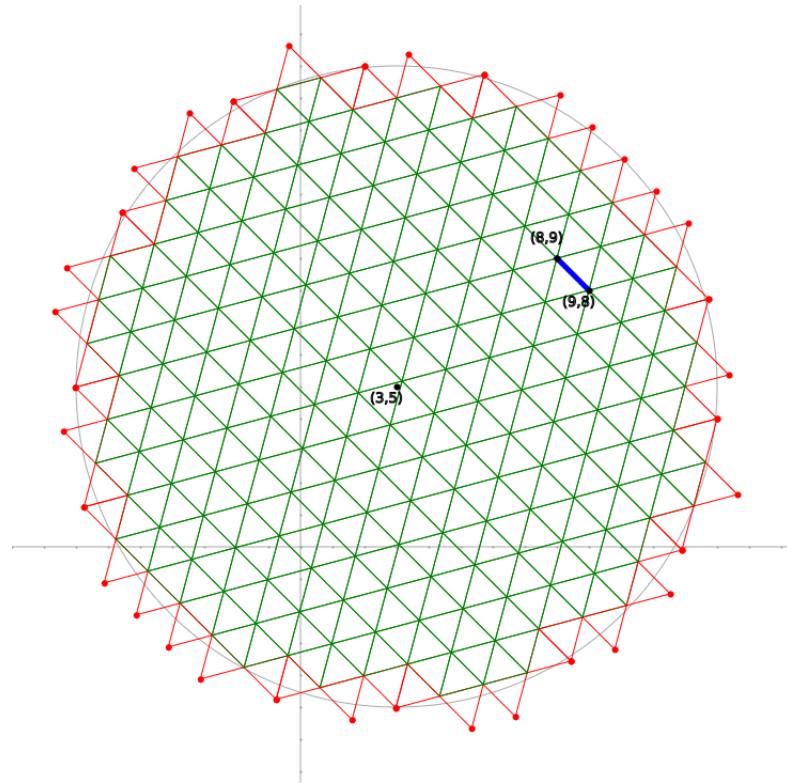
Como error de precisión use $epsilon = 10^{-6}$

Procurando ser exhaustivos, en este reto usaremos 2^8 casos de prueba

A continuación las figuras de los ejemplos:







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)



Han sido tiempos difíciles desde que el Ministerio de Magia cayó y **el-que-no-debe-ser-nombrado** tomó el control. Harry, Ron y Hermione se encuentran a salvo por ahora en la ancestral casa de la familia Black, el número 12 de Grimmauld Place, pero ellos necesitan continuar con la misión que Dumbledore les encomendó. Para esto, ellos necesitan moverse constantemente entre n lugares diferentes numerados de 1 a n , que incluyen casas de seguridad que la Orden del Fénix ha aprovisionado para ellos, así como lugares que tienen una fuerte conexión con el pasado del Señor Oscuro.

La situación empeora cuando ellos comienzan a sospechar que Harry aún tiene el rastro, lo cual les impide usar la aparición como su mecanismo de transporte. Debido a todo esto, ellos tienen que recurrir a los trasladadores como su mecanismo de viaje principal. Como todos sabemos, los trasladadores son objetos mágicamente encantados para transportar instantáneamente a cualquier persona que los toque a un destino específico (incluyendo su ubicación actual). Sin embargo, los trasladadores tienen una limitante y es que solo pueden ser usados una vez.

Los trasladadores son objetos bastante apetecidos y se encuentran regulados por el Ministerio, así que el trío dorado ha tenido que recurrir al comerciante de objetos mágicos Mundungus-Fletcher. Mundungus puede ofrecerle al grupo, por una módica cantidad de dinero, trasladadores básicos desde un punto u a un destino v , pero, dado que el trío no confía del todo en el famoso ladrón ellos pueden optar por dos tipos de trasladadores mejorados también:

- Trasladores mejorados con origen fijo: Este trasladador puede ser usado para viajar desde un punto origen fijo u a cualquier destino en el rango $[l, r]$ dado por Mundungus.
- Trasladores mejorados con destino fijo: Este trasladador puede ser usado para viajar de cualquier punto origen en el rango $[l, r]$ (dado por Mundungus), a un destino fijo u .

El trío no está muy seguro de cuál será su siguiente movimiento, pero ellos quieren estar preparados para cuando consigan una pista y tengan que moverse. Para esto, ellos quieren conocer cuál es la mínima



cantidad de dinero que deben gastar en trasladadores para llegar de Grimmauld Place a cualquier otro destino (incluyendo Grimmauld Place).

Input

La primera linea de las entrada contiene tres 3 enteros n , p y g ($1 \leq n, p \leq 10^5$; $1 \leq g \leq n$):

- n : Número de lugares a los que el trío necesita viajar,
- p : Número de opciones de trasladadores ofrecidos por Mundungus y
- g : posición de Grimmauld Place en la lista de lugares.

Las siguientes p líneas contienen la descripción de los trasladadores. Cada linea comienza con un string s , que define el tipo de trasladador (*basic*, *fixed – origin*, *fixed – destination*). Si el trasladador es de tipo *basic*, entonces seguirán tres 3 enteros v , u y w donde w es el costo de ese trasladador ($1 \leq v, u \leq n$, $1 \leq w \leq 10^9$). De lo contrario, seguirán cuatro 4 enteros u , l , r y w donde w es el costo de ese trasladador ($1 \leq u \leq n$, $1 \leq l \leq r \leq n$, $1 \leq w \leq 10^9$).

Output

Imprima una única línea con n números enteros separados por un espacio. El i -ésimo número es el costo para ir desde Grimmauld Place al i -ésimo destino, o -1 si es imposible llegar.

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) quiere comprar una nueva consola, pero tiene un presupuesto limitado y le gusta mucho la variedad. Hay una buena cantidad de consolas en el mercado y cada una de ellas corre una cantidad limitada de juegos. Lo que Iván desea es saber la máxima cantidad de juegos que pudiera tener comprando una sola consola que se ajuste a su presupuesto.

Input

La entrada consiste en varios casos de prueba. La primera línea contiene un número entero positivo T ($1 \leq T \leq 100$) el cual es el número de casos de prueba. Cada caso de prueba comienza con una línea que contiene dos números enteros positivos separados por espacio: $C \ B$ ($1 \leq C \leq 10^4$, $1 \leq B \leq 10^6$), los cuales significan respectivamente la cantidad de consolas disponibles en el mercado y la cantidad de dinero que Iván tiene como presupuesto para la compra de la consola. Luego se presentan C líneas, cada una de ellas conteniendo dos números enteros positivos separados por espacio: $C_i \ G_i$ ($1 \leq C_i \leq 10^5$, $1 \leq G_i \leq 10^4$), indicando respectivamente el costo y la cantidad de juegos que tiene la consola.

Output

For each test case, print on one line the maximum number of games Ivan could play on the console that fits your budget.

Examples

Input	Output
1	
3 10	
14 100	
9 15	
8 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) tiene ahora su consola, pero tiene que escoger de nuevo, tiene un presupuesto limitado y quiere maximizar la popularidad de sus juegos. Él cuenta con estadísticas de cuántas veces un juego ha sido comprado, por lo tanto, de cuán popular es el juego.

Ivan quiere saber cuánta “popularidad” puede obtener con su presupuesto. Él mide la popularidad sumando todas las ventas que sus juegos han tenido.

Input

La entrada es un solo caso de prueba. La primer línea tiene dos enteros: G ($1 \leq G \leq 10^3$) que denota el número de juegos que Ivan pudiera comprar, y el número B ($1 \leq B \leq 10^4$), que denota su presupuesto. Las siguientes G líneas tienen dos enteros, P ($1 \leq P \leq 10^5$) y S ($1 \leq S \leq 10^8$), denotando el precio y el número de ventas de un juego respectivamente.

Output

Un número solo indicando el máximo de popularidad que Ivan podría obtener.

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: 1 second / 256 megabytes (details on page 7)
Author(s): Wilmer-Emiro Castrillón-Calderón - U. de la Amazonía (Graduate)

Ha ocurrido una emergencia: tu amigo Manuel ha sido secuestrado y está encerrado en un laberinto, en el cual se puede mover en cuatro direcciones: arriba, abajo, a la izquierda y a la derecha, representadas por las letras *U*, *D*, *L* and *R*, respectivamente. Él necesita encontrar la manera de salir, pero el laberinto contiene muchas minas, es muy peligroso! Ahora tu debes rescatarlo.

Afortunadamente tu amiga Diana encontró un mapa del laberinto y descubrió que la única ruta segura es el camino más corto y más grande lexicográficamente. Ahora tu debes escribir un programa para encontrar la ruta segura. ¿Podrías ayudarlo?

Input

La entrada comienza con $T \leq 100$, el número de casos de prueba. Por cada caso de prueba hay una primera línea con números enteros positivos R y C ($2 \leq R, C \leq 500$) que indican la cantidad de filas y columnas del laberinto. Cada una de las siguientes R líneas contienen C caracteres que describen el laberinto. Los caracteres pueden ser los siguientes: ‘S’ la posición inicial donde se encuentra tu amigo, ‘X’ la salida, ‘#’ un muro (que no se puede atravesar) and ‘.’ un espacio libre (tu amigo puede caminar por estos espacios). El carácter ‘S’ y ‘X’ solo aparece una vez en el laberinto.

Output

Por cada caso de prueba imprima una línea con la ruta segura. Si no existe una ruta segura, entonces imprima **No exit**.

Examples

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

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 es un perro loco que se la pasa corriendo y ladrando por la casa todo el día.

La casa de *Lucky* puede ser vista como un laberinto, es decir, como una cuadrícula (una matriz) de celdas. Una celda puede ser de pared (la cual no se puede atravesar) o puede ser de piso (es decir, una celda por la cual *Lucky* puede correr). En cada momento *Lucky* puede moverse a una nueva celda desde su posición actual si ellas comparten un lado y las dos son celdas de piso.

Lucky está interesado en saber por cuantas celdas de piso el puede correr y ladrar como loco tomando como punto de partida una celda de piso. Obviamente *Lucky* esta muy ocupado todo el día cumpliendo con sus deberes, por este motivo el necesita de tu ayuda para resolver este reto.



Lucky is a crazy dog!

Input

La entrada comienza con un entero positivo T ($1 \leq T \leq 10$), denotando el número de casos de prueba.

Cada caso de prueba comienza con una línea que contiene dos números enteros positivos W H (Wide (ancho), High (alto), $3 \leq W, H \leq 1000$). El caso de prueba continua con H líneas, cada una de las cuales contiene W caracteres. Cada carácter representa el estatus de una celda en el laberinto como sigue:

1. ‘.’ - para indicar que es una celda de piso (celda por la cual *Lucky* puede correr),
2. ‘#’ - para indicar que es una celda de pared.

El caso de prueba continua con una línea que contiene un número entero positivo Q ($1 \leq Q \leq \min(W \cdot H, 10^4)$), el cual representa el total de celdas que se van a consultar. Por último se presentan Q líneas, cada una de ellas conteniendo dos números enteros positivos R C ($1 \leq R \leq H$, $1 \leq C \leq W$), los cuales representan la fila (row) y la columna (column) de la celda desde la cual *Lucky* quiere saber cuantas celdas se alcanzan (incluyendo la celda que se utiliza como punto de partida).

Output

Para cada caso de prueba, imprima una primera línea con el siguiente formato **Case idCase:**, donde **idCase** se reemplaza por el número del caso de prueba en el cual se encuentra, luego imprima Q líneas cada una de ellas conteniendo el total de celdas que son alcanzadas por *Lucky* para el punto de partida en el cual comienza en la consulta. Para mayor claridad en el formato de entrada y salida mirar los ejemplos a continuación.

Examples

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

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)



Un compuesto químico es una sustancia formada por la combinación de dos o más elementos de la tabla periódica. Los compuestos son representados por una fórmula química. Por ejemplo, el agua (H_2O) está constituida por dos átomos de hidrógeno y uno de oxígeno. Los elementos de un compuesto no se pueden dividir o separar por procesos físicos (decantación, filtración, destilación), sólo mediante procesos químicos. A la mínima unidad de un compuesto químico (esa misma que no se puede separar físicamente) se le llama molécula. Las moléculas son entidades formadas por átomos que tienen enlaces fuertes entre ellos y que las hacen suficientemente estables.

Una mezcla es un material formado por dos o más compuestos químicos unidos, pero no combinados químicamente. En una mezcla no ocurre una reacción química y cada uno de sus componentes mantiene su identidad y propiedades químicas. Algunas mezclas se pueden separar en sus componentes mediante procesos físicos (mecánicos o térmicos), como destilación, disolución, separación magnética, flotación, tamizado, filtración, decantación o centrifugación.

Una mezcla puede ser homogénea o heterogénea. Las mezclas homogéneas son aquellas en las que las propiedades intensivas son las mismas en toda la mezcla; es de apariencia uniforme y tiene las mismas propiedades en su conjunto (por ejemplo, sal disuelta en agua). En las mezclas homogéneas sus compuestos NO pueden diferenciarse a simple vista. Las mezclas heterogéneas son aquellas en las que las partes mantienen propiedades intensivas diferentes (por ejemplo, arena mezclada con aserrín). Las mezclas heterogéneas poseen una composición no uniforme en la cual se pueden distinguir a simple vista sus componentes, está formada por dos o más sustancias físicamente distintas, distribuidas en forma desigual.

El profesor Adenalleva Oibaf ha realizado observaciones a sustancias interesantes, ha obtenido información de los átomos que las componen y los enlaces que unen esos átomos con otros en la misma molécula. Ya que una sustancia puede ser un compuesto puro (es decir, formados por un solo compuesto) o una mezcla (es decir, formada por más de un compuesto), el profesor está interesado en conocer de cuantos componentes químicos están formadas cada una de las sustancias observadas.

Input

La primera línea tiene un número $n < 100$ con la cantidad de sustancias. Luego siguen n sustancias cuya descripción tiene varias líneas así: la primera línea de cada sustancia dos valores enteros $1 < a < 1000$ y $0 < e < 10000$, separados por un espacio, con la cantidad de átomos encontrados en la sustancia y la cantidad de enlaces entre esos átomos, respectivamente. Luego, para el caso de prueba siguen e líneas cada una con un par de enteros x y , separados por espacio, indicando que ese par de átomos (designados por enteros de 0 a $a - 1$) tienen un enlace entre sí.

Output

Para cada caso, su programa debe imprimir n líneas, una por cada sustancia, en la que diga: “Pure Compound!” si solo hay un compuesto químico en la sustancia, y “Mixture: m ”, si la sustancia tiene más de un compuesto, siendo m , la cantidad de compuestos de dicha sustancia.

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