Contents

1 I Training Camp de los Santanderes UFPS - Pragma - 2022

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 pensamiento computacional, resolución de problemas, programación de computadores, trabajo colaborativo y liderazgo.

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 ocho (8) años creciendo en Programación Competitiva junto a muchas universidades en varios países de toda Latinoamerica.

Otro actor importante ha sido el sector productivo. Cada año se ha vinculado una empresa de la región que aporta alguna ayuda al evento. Para este año 2022 se vinculó la empresa Pragma. Y, considerando la experiencia de la VIII Maratón Interna UFPS 2022, la empresa Pragma decidió patrocinar un entrenamiento y maratón con las universidades de los santanderes, buscando motivar y apoyar el crecimiento de la Programación Competitiva en la región. De manera ágil, usando un grupo de WhatsApp, se organizó con los coach de varias universidades, con Pragma y con RPC y se acordó que la última fecha del año de RPC coincidera con el entrenamiento.

Es así como esta competencia es el cierre de un evento de entrenamiento liderado por RPC, UFPS y Pragma. Nos complace presentar un conjunto de problemas inéditos, escritos por estudiantes, profesores y graduados.

Reconocimiento y agradecimiento especial al equipo de trabajo de todos los años:

- Hugo Humberto Morales Peña - Profesor Universidad Tecnológica de Pereira, Colombia
- Milton Jesús Vera Contreras - Profesor UFPS (desde Cúcuta)
- Eddy Ramírez - Profesor UNA & TEC, Costa Rica
- Gabriel Gutiérrez Tamayo - UTP
- Carlos Andres Arias - UTP
- Jefferson Ferney Jaramillo Vega - Pragma (desde Cúcuta)
- Diana Espinosa y su semillero - Profesora Universidad de la Amazonía, Colombia
- Equipo Red de Programación Competitiva (Diana Espinosa, Fabio Avellaneda, Johany Careño)
- Compañeros y Directivos de la UFPS Cúcuta
- Universidades participantes: De Bucaramanga-Santander, Universidad Industrial de Santander UIS y Universidad de Investigación y Desarrollo UDI; de Florencia-Caquetá Universidad de la Amazonía; de Villa del Rosario, Universidad de Pamplona; de Ocaña Universidad Francisco de Paula Santander y de Cúcuta Universidad de Santander, UDES, Universidad Remington, Universidad Simón Bolívar y UFPS Cúcuta.

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 2015, logrando por 7 años consecutivos la clasificación a fase Regional Latinoamericana. En las dos competencias la UFPS se ha logrado ubicar en el top 10.

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. 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.
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.

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 ubicarse en el top 3 de la competencia. 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 ;)

6 Fe de Erratas

Errare humanum est, lo importante es reconocer y corregir el error. Si durante esta competencia se presentan errores, aparecerán aquí y se actualizarán en linea, informando a través de la plataforma:

1. En el problema L, la cota máxima era $1n10^5$. En los casos de prueba, por error, se superó esa cota. Se ajustaron los casos de prueba y se recalificó, eliminando la penalidad.
2. En el problema A, la explicación del output dice "Felicidades Bine, listos para elecciones." y en los ejemplos dice "Felicidades Bine, listo para elecciones.". Se ajustó el cuadernillo.
3. Debido al alto volumen de trabajo presenciales en Cúcuta, donde el cuadernillo estaba impreso en papel, tardamos en subir el cuadernillo digital.

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 ocho años entre la UFPS y RPC, siempre con mucho entusiasmo y cariño para mantener viva la llama de la Programación Competitiva.

Problem A. Amigos

Source file name:	Amigos.c, Amigos.cpp, Amigos.java, Amigos.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Milton Jesús Vera Contreras UFPS (Professor)`

El señor Bine es un gran político y usa sus redes sociales para ampliar su alcance y conseguir mayorías demoledoras en las elecciones. Estuvo analizando en detalle su lista de amigos en redes sociales e identificó que sus amigos se pueden clasificar en grupos que no se relacionan entre si. Esto es problemático, porque en tiempos electorales uno de esos grupos podría retirarle el apoyo. Lo mejor es siempre tener un único grupo homogéneo de amigos.

Para resolverlo, Bine decidió hacer una fiesta especial, invitando un amigo de cada uno de sus grupos de amigos. La estrategia de Bine es que, en medio de copas, comida, baile y tertulia, sus amigos se relacionen en sus redes sociales, pasando de varios grupos a un grupo único de prosélitos.

Bine no logra decidir a quienes invitar de cada grupo de amigos, duda entre el amigo más antiguo, el más reciente, el más fiel o el más carismático... Su asesor tecnológico le recomendó hacer listas ordenadas alfabéticamente e invitar siempre al que aparezca primero en la lista, para reducir un poco los sesgos y fortalecer la estrategia. Además, como todo Ingeniero asesor de políticos, le recomendó contratar un ingeniero que sepa del tema, para que le ayude a implementar un modelo computacional que resuelva el problema.

¿Podrías ayudarle al señor Bine?

Input

Un número $1 <= T <= 100$ con el total de casos de prueba. Cada caso T_i inicia con dos números $3 <= n <= 100$ y $3 <= m <= 100$ separados por espacio en blanco. El número n corresponde al total de amigos del señor Bine y el número m corresponde al total de relaciones en redes sociales entre parejas de amigos del señor Bine.

Después sigue una línea con una lista de n nombres separados por espacio en blanco: $\{nombre_1, nombre_2, nombre_3, ...nombre_n\}$, el nombre de cada uno de los amigos de Bine, sin espacios en blanco.

Y, finalmente, m líneas con la lista de parejas de nombres, a b separadas por espacio en blanco, cada pareja en una línea, indicando que la persona a se relaciona en redes sociales con la persona b. Tanto a

como b están en la lista de nombres de amigos de Bine y nunca hay nombres repetidos. Lógicamente el señor Bine es excluido del modelo.

Output

Si los amigos de Bine conforman un único grupo homogéneo, sin diferencias, imprimir un aviso *"Felicidades Bine, listos para elecciones."*. En caso contrario, imprimir un aviso *"Invitados a la fiesta de Bine:"*, seguido de la lista de amigos que Bine debe invitar para que se relacionen en redes sociales y así tener un único grupo homogéneo de amigos. La lista de amigos debe estar en una única línea, separados por espacio en blanco.

Examples

Input	Output
2 6 6 Luis Maria Carmen Juan Pedro Lucas Lucas Pedro Juan Pedro Maria Luis Luis Carmen Pedro Maria Carmen Lucas 6 4 Luis Maria Carmen Juan Pedro Lucas Lucas Pedro Juan Pedro Maria Luis Luis Carmen	Felicidades Bine, listos para elecciones. Invitados a la fiesta de Bine: Carmen Juan

Problem B. Batalla por Medallas

Source file name:	Batalla.c, Batalla.cpp, Batalla.java, Batalla.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Eddy Ramírez - UNA & TEC Costa Rica (Professor)`

Una organización desea realizar una maratón en la cual se cuenta con N personas inscritas. Como parte de la métrica en este tipo de eventos, a todos los participantes que concluyan la maratón se les otorgará una medalla, por lo que necesitan comprar un mínimo de N medallas. El problema al que se enfrentan es que el proveedor local sólo vende paquetes de cierta cantidad de medallas, por cierto precio.

El proveedor ofrece P diferentes paquetes. La idea de la organización consiste en comprar los paquetes que sumen al menos N medallas, pero cuyo costo sea el menor. Un paquete de medallas puede ser comprado más de una vez.

Input

La entrada consiste en un caso de prueba que consta de tres líneas:

La primera línea contiene dos enteros N y P separados por espacio. Con $1 \leq N \leq 10^6$ indicando el número de personas registradas para la maratón. Y P, con $1 \leq P \leq 50$ indicando el número de paquetes que vende el proveedor. La segunda línea contiene P enteros separados por espacio. Esta lista es la cantidad M de medallas que se venden en cada paquete, con $M_i > M_{i+1}$ y $1 \leq M_j \leq 10^6$ La tercera línea contiene P enteros separados por espacio. Esta es la lista de precios de cada grupo de medallas. $1 \leq X_i \leq 10^6$

Output

La salida consiste en una línea con un único entero, indicando el mínimo costo para poder comprar al menos N medallas.

Examples

Input	Output
15 3 12 8 4 45 30 16	60
60 3 40 30 20 80 65 45	125

Problem C. Cien Años de Soledad

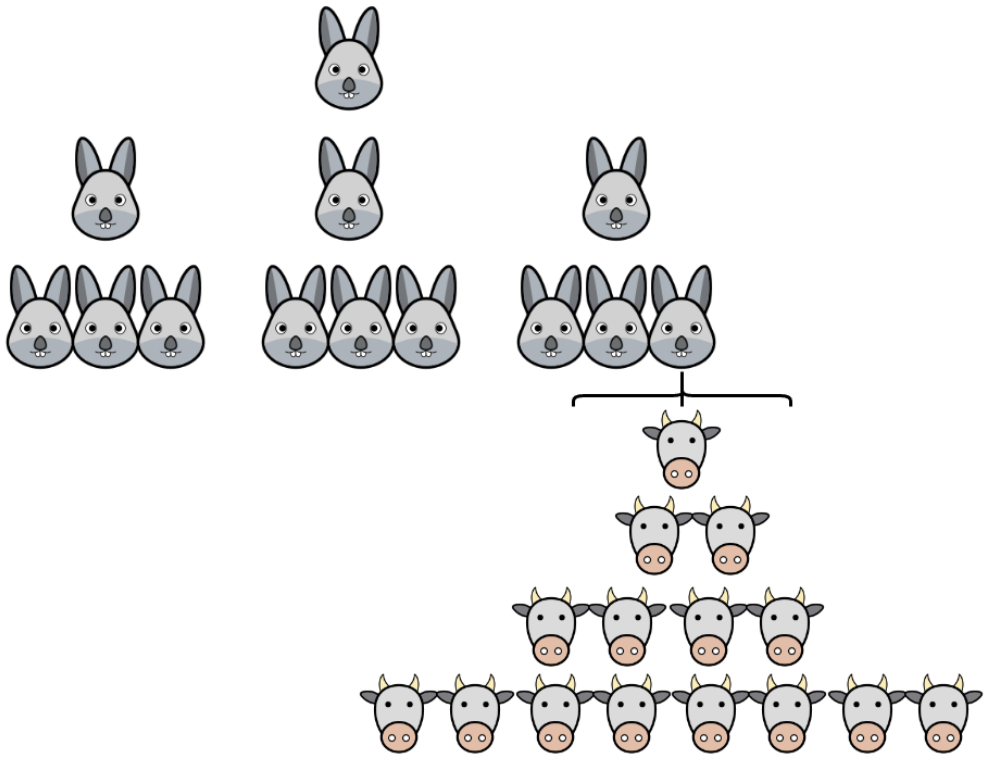
Source file name:	Cien.c, Cien.cpp, Cien.java, Cien.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Milton Jesús Vera Contreras UFPS (Professor)`

Fragmento de Cien Años de Soledad (Gabriel García Márquez)

Fue en esa época que le dio a Petra Cotes por rifar conejos. Se reproducían y se volvían adultos con tanta rapidez, que apenas daban tiempo para vender los números de la rifa. Al principio, Aureliano Segundo no advirtió las alarmantes proporciones de la proliferación. Pero una noche, cuando ya nadie en el pueblo quería oír hablar de las rifas de conejos, sintió un estruendo en la pared del patio. «No te asustes -dijo Petra Cotes-. Son los conejos.» No pudieron dormir más, atormentados por el tráfago de los animales. Al amanecer, Aureliano Segundo abrió la puerta y vio el patio empedrado de conejos, azules en el resplandor del alba. Petra Cotes, muerta de risa, no resistió la tentación de hacerle una broma. -Estos son los que nacieron anoche -dijo. -¡Qué horror! -dijo él-. ¿Por qué no pruebas con vacas? Pocos días después, tratando de desahogar su patio, Petra Cotes cambió los conejos por una vaca, que dos meses más tarde parió trillizos. Así empezaron las cosas. De la noche a la mañana, Aureliano Segundo se hizo dueño de tierras y ganados, y apenas si tenía tiempo de ensanchar las caballerizas y pocilgas desbordadas. Era una prosperidad de delirio que a él mismo le causaba risa, y no podía menos que asumir actitudes extravagantes para descargar su buen humor. «Apártense, vacas, que la vida es corta»

Un poco cansados de resolver problemas de programación, los estudiantes del semillero comenzaron a profundizar en la obra del premio Nobel de Literatura. Leyendo el pasaje anterior, revivió la pasión por la computación y comenzaron a cuestionarse sobre cuántos conejos y cuántas vacas llegaron a tener Petra Cotes y Aureliano Segundo.

De acuerdo a lo que investigaron, contrario a lo que dice la obra del Nobel, el comportamiento de reproducción de los animales en Macondo y el modelo de negocio de Petra Cotes seguían el siguiente patrón:



1. Los conejos se triplicaban cada mes. Los conejos recién nacidos eran conservados y los progenitores eran vendidos. Inicialmente solo tenían una coneja que llegó a Macondo preñada.
2. Cuando Aureliano Segundo se aburría de los conejos, Petra Cotes los vendía y compraba vacas. Como eran conejos conjurados por Melquiades, cada conejo alcanzaba para comprar una vaca y sobraba dinero.
3. Las vacas se duplicaban cada año. Las vacas recién nacidas eran conservadas y sus progenitores eran vendidos.
4. Cuando Petra Cotes se cansaba de las vacas, Aureliano Segundo las vendía y compraba conejos.
5. Como todo en Macondo, el ciclo se repetía, tal como el coronel y sus guerras y pescaditos de oro...

Según documentos históricos de Macondo, existieron varios ciclo de conejos y vacas, Cada ciclo de conejos duró a meses y le siguió un ciclo de vacas que duró b años. Esos números a y b fueron borrados accidentamente, pero en los pergaminos de Melquidades aparecen los números n con el total de vacas que había en Macondo para aquella época, la más fecunda de su historia. El número está en sistema de numeración binario y tiene más de 10^6 dígitos.

Algunos estudiantes han intentado calcular con Python los números a y b, porque dicen que Python es rápido para procesar números grandes. Pero no lo han conseguido... ¿Podrías ayudarles en otro lenguaje de programación o enseñarles cómo hacerlo en Python?

Input

Varios casos de prueba, cada caso se prueba independientemente. Cada caso de prueba consiste en un número n en sistema de numeración binario con longitud $10^6 <= d < 10^{10}$ dígitos binarios (1 y 0). Se garantiza que el número n cumple con el patrón de Cien Años de Soledad.

Output

Por cada caso de prueba imprima una línea con la cantidad de años b y de meses a, con el siguiente formato: *b años de vacas y a meses de conejos*

Examples

Input	Output
11011000	3 años de vacas y 3 meses de conejos
1001000	3 años de vacas y 2 meses de conejos

Note

Intentamos poner en los ejemplos de entrada y salida una copia de los pergaminos de Melquiades con el número n, pero por tamaño no fue posible. Confiamos en que los ejemplos sean de utilidad ;) ...

Use Fast I/O

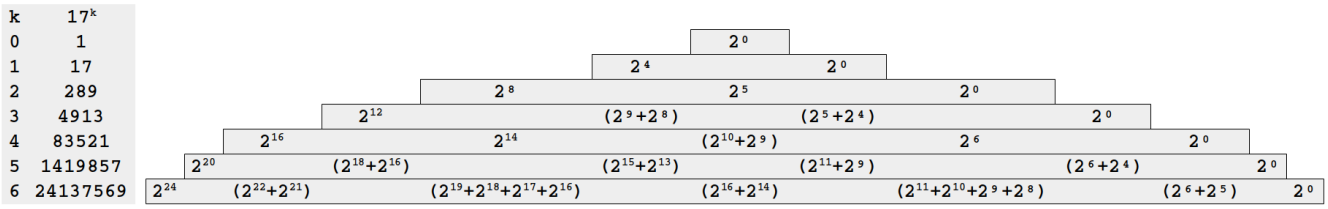
Problem D. Dos y dos son cuatro, cuatro y dos son son seis..

Source file name:	Dos.c, Dos.cpp, Dos.java, Dos.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Milton Jesús Vera Contreras UFPS (Professor)`

"Dos y dos son cuatro, cuatro y dos son seis,
seis y dos son ocho, y ocho dieciséis,
y ocho veinticuatro, y ocho treinta y dos.
Estas son las cuentas que he sacado yo"

Aunque los estudiantes aman la programación de computadores, se les dificulta realizar cálculos mentalmente y hacer análisis aritmético y algebraico de una situación. Además, odian los problemas donde falta información y deben completar patrones. Para ayudar a sus estudiantes a cultivar esas habilidades, que son propias del Pensamiento Computacional y son muy demandadas en el mercado, al profesor le gusta proponerles retos donde se desafía el pensamiento y las habilidades de programación.

En este caso el reto consiste en calcular la potencia entera k de un número entero p (p^k) y luego imprimir el resultado como sumas de potencias de 2, como se ilustra en la imagen para 17^k con k desde 0 hasta 6:



Cualquier número entero p^k se puede representar con el patrón de la figura anterior, pero resulta complejo generalizar. Entonces, el profesor consideró solo algunos casos, los más sencillos y 17 es uno de ellos, en cambio, ni 14, 15, 16, 18 ni 19 cumplen el patrón adoptado por el profesor.

El profesor espera que los estudiantes resuelvan el reto solo con esta información, para lo cual deben, primero, identificar los casos que el profesor consideró y, después de eso, resolver el reto.

Input

Un número $0 < T <= 100$ con la cantidad de casos de prueba. Cada caso de prueba consiste en una pareja de números enteros separados por espacio en blanco p k. Se cumple que $3 <= p <= 10^{10}$ y $3 <= k <= 100$.

Output

Si los datos de entrada no cumplen el patrón descrito por el profesor, imprima un aviso:
"¡Profe: sea serio!"

Si los datos de entrada cumplen el patrón descrito por el profesor, imprima en una línea el patrón de potencias de 2 que representa p^k, siguiendo estas reglas de impresión, tal como se ilustra en la imagen del enunciado y en los ejemplos de entradas y salidas:

1. Represente las potencias con el caracter circunflejo ^, por ejemplo 2^3.
2. Cuando hay suma de más de una potencia, use el operador +, agrupe en paréntesis y no deje espacios en blanco.

3. Para separar cada suma de potencias, use el caracter coma seguido de un espacio en blanco ", ".

Examples

Input
3 17 3 17 6 14 5
Output
2^12, (2^9+2^8), (2^5+2^4), 2^0 2^24, (2^22+2^21), (2^19+2^18+2^17+2^16), (2^16+2^14), (2^11+2^10+2^9+2^8), (2^6+2^5), 2^0 ¡Profe: sea serio!

Note

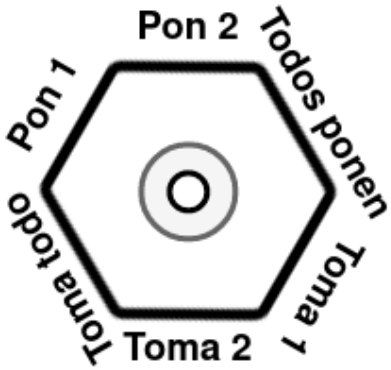
Use Fast I/O

Problem E. Enredados con la Pirinola

Source file name:	Enredados.c, Enredados.cpp, Enredados.java, Enredados.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Milton Jesús Vera Contreras UFPS (Professor)`

Un casino quiere implementar un juego de pirinola usando Internet de las cosas, pero están enredados en el diseño y decidieron contratar a los estudiantes que asistieron al *Primer Training Camp de los Santanderes*, para que ayuden a desarrollar todo el software requerido para el proyecto.

El juego de pirinola consiste en seis (6) jugadores que giran un cilindro hexagonal, cuyos lados están etiquetados con seis acciones: "Toma todo", "Pon 1", "Pon 2", "Todos ponen", "Toma 1" y "Toma 2". Si se mira la pirinola desde arriba, mientras gira, su forma es hexagonal, como se ilustra en la figura.



La mesa donde se juega es hexagonal, igual que la pirinola y los jugadores se ubican en cada lado. Cada jugador se ubica inicialmente en uno de los seis lados, el que esté disponible y el jugador prefiera. Cada jugador se enumera de 1 a 6 desde la posición "Toma todo", en el sentido de las manecillas del reloj. Se asume siempre como posición inicial del juego "Toma todo", que es la más favorable.

Por turnos, cada jugador hace girar la pirinola y según el lado de la pirinola que quede frente al jugador será la acción que este debe realizar: Si sale "Toma 1" o "Toma 2", el jugador puede tomar uno o dos dólares. Si sale "Pon 1" o "Pon 2", el jugador debe poner uno o dos dólares. Si sale "Toma todo", el jugador puede tomar la mitad de todos los dólares sobre la mesa, porque el casino nunca pierde. Y si sale "Todos

ponen", todos los jugadores ponen en la mesa un dólar. Si sale "Pon 2" y el jugador solo tiene 1 dolar, el casino pierde el otro dolar y el jugador deja de jugar.

Puesto que la pirinola usa Internet de las cosas, el jugador que tiene el turno pulsa un botón y el sistema del casino calcula un número aleatorio $10^3 <= n <= 10^9$, que corresponde a la cantidad de movimientos que hace la pirinola y un número aleatorios $1 <= \theta <= 360$, que corresponde al ángulo del movimiento de la pirinola, en el sentido de las manecillas del reloj. Después de los n giros de $\theta°$ la pirinola se detiene.

Si los lados de la mesa de juego no son totalmente paralelos a los lados de la pirinola, se considera mala suerte y todos los jugadores deben poner un dolar, para alejar la mala suerte. El jugador que tuvo mala suerte agotó su turno y se da paso al siguiente jugador. En este caso la pirinola mantiene la posición antes del último lanzamiento. Si los lados de la mesa de juego son paralelos, se sigue el juego.

El casino está interesado en identificar y analizar el comportamiento de la pirinola para determinada cantidad de movimientos T. Y para eso recurrió al *Primer Training Camp de los Santanderes*, pues está seguro que todos los equipos propondrán soluciones creativas y eficientes.

Input

La primera linea contiene un número e, que corresponde al total de casos de prueba. Cada caso de prueba inicia con un número $10 <= T < 5000$, el total de jugadas. Luego siguen T líneas, cada línea tiene una pareja de números n y $\theta°$, que corresponden a la cantidad de movimientos y el ángulo de esos movimientos. Se cumple siempre que $10^3 <= n <= 10^9$ y $1 <= \theta <= 360$

Output

Por cada caso de prueba se requiere imprimir siete líneas, una línea por cada jugador y una linea al final con "*". Cada línea, correspondiente a cada jugador, debe tener ocho números $t, m, l1, l2, l3, l4, l5, l6$ separados por espacio en blanco. Estos números corresponden, respectivamente, al total de lanzamientos de la pirinola t del jugador, la cantidad de lanzamientos que fueron de mala suerte para ese jugador y la cantidad de lanzamientos en los que ese jugador obtuvo cada uno de los lados de la pirinola: "Toma todo", "Pon 1", "Pon 2", "Todos ponen", "Toma 1" y "Toma 2".

Examples

Input	Output
2 6 1 60 2 60 3 60 4 60 5 60 6 60 6 12 45 3 60 15 71 8 30 6 20 60 11	1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 * 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 *

Note

Use Fast I/O

Problem F. Toby el Fontanero

Source file name:	Fontanero.c, Fontanero.cpp, Fontanero.java, Fontanero.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Carlos Andres Arias - UTP`

En esta ocasión Toby tiene una cantidad de tubos, los cuales son descritos como una pareja $\{a,\ b\}$ donde a, b son el diámetro inicial y final respectivamente, el largo de los tubos no nos interesa. Como nuestro amigo es muy caprichoso el no desea rotar ningún tubo, es decir, si el tubo fue dado como $\{a,\ b\}$ entonces el no podrá rotarlo de tal forma que quede $\{b,\ a\}$.

Él desea conectar **todos** estos tubos tal que sí el tubo A está conectado con el tubo B, entonces, el diámetro final de A es igual al diámetro inicial de B.

Por ejemplo, si se tienen los tubos $A = \{2,\ 4\}$ y $B = \{5,\ 2\}$ en este caso sólo podríamos conectar $B \rightarrow A$, pero no $A \rightarrow B$, ya que el diámetro final de A es diferente al diámetro inicial de B.

Nuestro amigo tiene el poder de cambiar **a lo sumo una vez** el diámetro (inicial o final) de cualquier tubo.

Toby es muy perezoso y quiere que tú hagas el trabajo por el, entonces lo que debes hacer es decir si es posible o no conectar las N tuberías usando o no el poder.

Ejemplo:

Dados los tubos $[\{4,\ 5\},\ \{3,\ 4\},\ \{2,\ 3\},\ \{4,\ 8\}]$ sin utilizar el poder es imposible conectar los cuatro tubos, pero si usamos el poder y cambiamos el diámetro final del cuarto tubo tal que $\{4,\ 8\} \rightarrow \{4,\ 4\}$, podríamos conectar los tubos de la siguiente manera $[\{2,\ 3\},\ \{3,\ 4\},\ \{4,\ 4\},\ \{4,\ 5\}]$.

Input

La entrada consta de varios casos de prueba y en cada uno de ellos deberás de leer un entero N $(1 \leq N \leq 12)$ que indicará el número de tubos, después debes de leer N lineas que tendrán dos enteros a y b $(1 \leq a,\ b \leq 10)$ los cuales serán el diámetro inicial y final respectivamente del i-ésimo tubo.

Se debe leer hasta el fin de archivo.

Output

Para cada caso de prueba, debes imprimir "`si`" si es posible conectar los tubos dados en el caso usando o no el "poder", de lo contrario imprima "`no`"; sin las comillas.

Examples

Input	Output
4 4 5 3 4 2 3 4 8 3 1 1 2 2 3 3	si no

Problem G. GCD

Source file name:	GCD.c, GCD.cpp, GCD.java, GCD.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Gabriel Gutiérrez Tamayo - UTP`

Dado un vector usted debe contar el numero de sub-arrays que tengan máximo común divisor (Greatest Common Divisor - GCD) igual a 1.

Input

La primera linea de entrada contiene un entero N ($1 \leq N \leq 10^5$), en la siguiente linea hay N enteros a_i ($1 \leq a_i \leq 10^9$).

Output

Imprimir en una sola línea un número entero indicando la cantidad de sub-arrays con GCD igual a 1. Tener en cuenta que este resultado no siempre cabe en un número entero de 32 bits.

Examples

Input	Output
10 25 35 45 50 35 49 14 3 42 15	29

Problem H. Humbertov y los Números Poligonales

Source file name:	Humbertov.c, Humbertov.cpp, Humbertov.java, Humbertov.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Hugo Humberto Morales Peña - UTP`

Ya es bien conocido por todos que el profesor **Hubertov Moralov** tiene una rara obsesión con los números triangulares. En días pasados revisando en Wikipedia se encontró con lo siguiente:

En 1638, Fermat propuso que cada número entero positivo es la suma de como máximo tres números triangulares, cuatro números cuadrados, cinco números pentagonales, y n números n-poligonales. Esto es lo que se conoce como el Teorema de los números poligonales de Fermat.

Apoyados en el **Teorema de los números poligonales de Fermat**, el profesor Moralov quiere contar la cantidad de formas diferentes de obtener un número entero positivo n con la suma de uno, dos o tres números triangulares, donde el número triangular que se suma podría llegar a ser el mismo. Para poder realizar esta tarea el profesor Moralov necesita de tu ayuda ya que el se ha dado cuenta que tu haz participado en el *Primer Training Camp de los Santanderes.*

Por ejemplo, el número 9 se obtiene de dos formas diferentes, sumando tres veces el número triangular 3 $(3+3+3=9)$ y sumando los números triangulares 3 y 6 $(3+6=6+3=9)$.

Input

La entrada del problema comienza con un número entero positivo t $(1 \leq t \leq 10^5)$, el cual representa el total de casos de prueba. Cada caso de prueba es presentado en una sola línea que contiene un número entero positivo n $(1 \leq n \leq 10^5)$.

Output

Para cada caso de prueba, su programa debe imprimir una línea con un número entero positivo, el cual denota el total de formas diferentes como se puede obtener el número entero positivo n por la suma de uno, dos o tres números triangulares. Cada caso de prueba debe generar una línea en la salida.

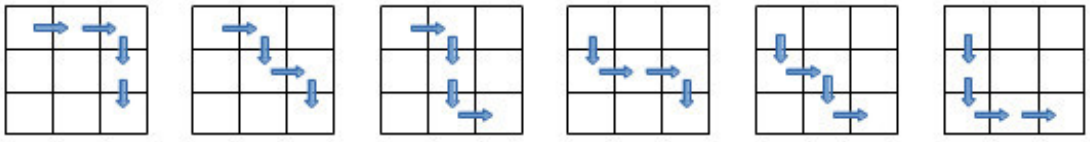
Examples

Input	Output
15 1 2 3 4 5 6 7 8 9 10 21 52 66 112 177	1 1 2 1 1 2 2 1 2 2 4 5 6 7 8

Problem I. Rutas I

Source file name:	RutasI.c, RutasI.cpp, RutasI.java, RutasI.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Gabriel Gutiérrez Tamayo - UTP`

Dado el tamaño de un cuadrado de lado N, se debe determinar la cantidad de formas de llegar desde la esquina superior izquierda hasta la esquina inferior derecha; solo se tienen 2 movimientos: mover una casilla hacia abajo o hacia la derecha. El siguiente ejemplo muestra las posibles rutas para un cuadrado de tamaño $N = 3$:



Input

La primera linea de entrada contiene un número entero positivo T ($1 \leq T \leq 10^3$) indicando el número de casos de prueba, cada una de las siguientes T lineas contiene un número entero positivo N ($2 \leq N \leq 10^3$).

Output

Para cada caso de prueba debe imprimir una linea con un número entero positivo indicando la cantidad de rutas posibles, como el número puede ser muy grande imprimirlo módulo $10^9 + 7$.

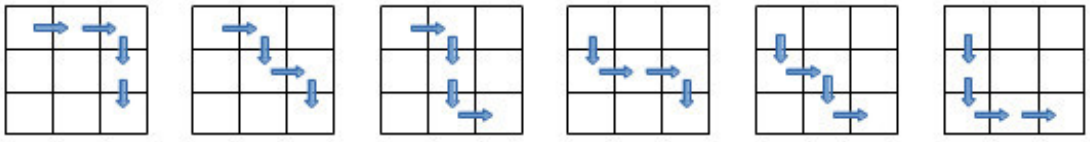
Examples

Input	Output
4 2 3 4 999	2 6 20 779828782

Problem J. Rutas J

Source file name:	RutasJ.c, RutasJ.cpp, RutasJ.java, RutasJ.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Gabriel Gutiérrez Tamayo - UTP`

Dado el tamaño de un cuadrado de lado N, se debe determinar la cantidad de formas de llegar desde la esquina superior izquierda hasta la esquina inferior derecha; solo se tienen 2 movimientos: mover una casilla hacia abajo o hacia la derecha. El siguiente ejemplo muestra las posibles rutas para un cuadrado de tamaño $N = 3$:



Input

La primera linea de entrada contiene un número entero positivo T ($1 \leq T \leq 10^5$) indicando el número de casos de prueba, cada una de las siguientes T lineas contiene un número entero positivo N ($2 \leq N \leq 10^5$).

Output

Para cada caso de prueba debe imprimir una linea con un número entero positivo indicando la cantidad de rutas posibles, como el número puede ser muy grande imprimirlo módulo 955049953.

Examples

Input	Output
4 2 3 4 10	2 6 20 48620

Problem K. Kripto y sus Galletas

Source file name:	Kripto.c, Kripto.cpp, Kripto.java, Kripto.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Eddy Ramírez - UNA & TEC Costa Rica (Professor)`

Kripto, el superperro, tiene una fijación por las galletas de supermantequilla. Superman, debió limitar la cantidad de galletas que podía comer por día y para ello le propuso un reto.

Dado dos números naturales n y p, siendo p primo y $n > 1$, Kripto debe indicar un número k tal que $\frac{n!}{p^k} \in \mathbb{N}$. Si se cumple la propiedad, Kripto recibe k galletas, pero si resulta que $\frac{n!}{p^k} \notin \mathbb{N}$, Kripto deja de recibir galletas ese día.

Ahora, Kripto es un perro muy inteligente y desea comer la máxima cantidad de galletas que pueda. Por ejemplo, si Superman le ofrece 100 y 5 como n y p respectivamente, Kripto puede decir cualquier valor k entre 0 y 24, esto porque $\forall k \in [0, 24](k \in \mathbb{N}$ hace que la división de 100! por 5^k sea entera. En este caso, Kripto escogería 24, porque le proporcionaría la máxima cantidad de galletas.

El problema que tiene Kripto es que es un perro y sus patas no le permiten escribir un programa que agilice este proceso, por eso le ha solicitado a usted que le escriba un programa que le permita resolver todos los retos que Superman le proponga de forma rápida.

Input

La entrada consiste en una primera línea D con $1 \leq D \leq 10^5$, indicando el número de días que Superman ha planeado. Las siguientes D líneas consisten en dos enteros separados por espacio, n y p, para todos los casos se cumple que $1 < n, p \leq 10^9$ y que p es primo, indicando el n y el p del día correspondiente.

Output

La salida es una línea con un número k, por cada día indicando el máximo número de galletas que Kripto puede pedir ese día.

Examples

Input	Output
2 100 5 1500000000 7	24 249999994

Problem L. Luces de Navidad

Source file name:	Luces.c, Luces.cpp, Luces.java, Luces.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Milton Jesús Vera Contreras UFPS (Professor)`

Durante su largo viaje por tierra hacia Cúcuta, desde Ocaña y Bucaramanga, para el Primer Training Camp de los Santanderes, los talentosos estudiantes de las Universidades UFPSO, UDI y UIS observaron atentos las luces navideñas. En una hoja de papel escribían el total de las luces n que contaban cada kilómetro k de carretera. Este es un fragmento del papel que tenían los estudiantes:

$$1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105...$$

Debido a que se dedicaron a estudiar Programación Competitiva, olvidaron el mecanismo de conteo que usaron, por lo cual decidieron escribir un programa de computador que calcule el kilómetro k correspondiente a esa cantidad de luces n.

Input

La primera linea de entrada contiene un número entero positivo T ($1 \leq T \leq 10^5$) indicando el número de casos de prueba. Cada una de las siguientes T lineas contiene un número entero positivo n ($1 \leq n \leq 10^5$), que corresponde al total de luces contadas en determinado kilómetro k.

Output

Para cada caso de prueba debe imprimir una linea con un número entero positivo k indicando el kilómetro en el cual se contaron las n luces.

Examples

Input	Output
3 15 28 45	5 7 9

Problem M. What is the k-th way to get M?

Source file name:	Mkth.c, Mkth.cpp, Mkth.java, Mkth.py
Input:	`standard input`
Output:	`standard output`
Author(s):	`Gabriel Gutiérrez Tamayo - UTP`

A positive integer M can be represented by a list of positive integers if the sum of the elements in that list is equal to M. Also, M can be represented by many lists, where two lists A and B are different if they have different size or if there is a position where they have different value.

A list A is lexicographically smaller than a list B if any of the following conditions hold:

- A is prefix of B and $|A| < |B|$
- There exists an integer i $(1 \leq i \leq min(|A|, |B|))$ such that $A_i < B_i$ and for each j $(1 \leq j < i)$ it holds that $A_j = B_j$

Here $|X|$ denotes the size of list X.

Your task is to find the k-th lexicographically smallest list among all the lists in which the sum of its elements is equal to M, but since for very small values of M there are already more than 2^{64} lists that can represent them, the k will be given as a hexadecimal string s.

Example: If M is 5, the first 7 ways to represent it as the sum of positive integers are:

$$5 = 1 + 1 + 1 + 1 + 1$$
$$5 = 1 + 1 + 1 + 2$$
$$5 = 1 + 1 + 2 + 1$$
$$5 = 1 + 1 + 3$$
$$5 = 1 + 2 + 1 + 1$$
$$5 = 1 + 2 + 2$$
$$5 = 1 + 3 + 1$$

Input

The first line contains an integer t $(1 \leq t \leq 10^5)$ indicating the number of test cases. Each test case consists of one line containing one integer M $(1 \leq M \leq 10^3)$ and one hexadecimal string s $(1 \leq |s| \leq 10^3)$.

Hexadecimal strings will be given using only the following characters: {`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`}

Output

For each test case, print `NO` if there are less than k lists that can represent M, otherwise print two lines, on the first print `YES` followed by an integer L indicating the number of elements in the k-th lexicographically smallest list and on the next line print the numbers from that list.

Examples

Input	Output
6 2 2 3 1 5 7 3 A 10 AC 17 CF5	YES 1 2 YES 3 1 1 1 YES 3 1 3 1 NO YES 5 1 2 2 2 3 YES 10 1 1 1 1 3 1 5 2 1 1