Concurso de Programación

IX AdaByron Madrid

http://www.ada-byron.es

Cuadernillo de problemas

accenture

inetum
Positive digital flow

instituto IMDEA software

POLITÉCNICA
UNIVERSIDAD POLITÉCNICA DE MADRID

acm UPM

Escuela Técnica Superior de Ingenieros Informáticos

11 de marzo de 2023

In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selections amongst them for the purposes of a calculating engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.
Ada Byron

11 de marzo de 2023

Índice

Autores de los problemas:

- Catalin Sorin Covaci (Universidad Politécnica de Madrid)
- Isaac Lozano Osorio (Universidad Rey Juan Carlos)
- Juan Céspedes Prieto (Instituto IMDEA Software)
- Manuel Carro Liñares (Universidad Politécnica de Madrid e Instituto IMDEA Software)
- Margarita Capretto (Instituto IMDEA Software)
- Santiago Tapia Fernández (Universidad Politécnica de Madrid)

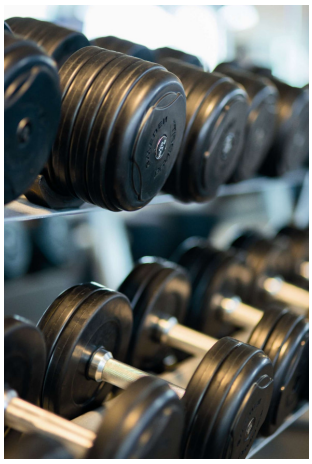
Revisión de los problemas:

- Jaime Fuertes Llorens (Universidad Politécnica de Madrid)
- Jorge Fernández Moreno (Universidad Politécnica de Madrid)

Tiempo: 2 segundos

A
El bruxeador

Un día Bruxo fue al gimnasio y, como si fuese el protagonista de una película, un señor se le acercó espontáneamente ofreciéndose a enseñarle boxeo y ser su entrenador. Desde ese día, Bruxo está decidido a ser el mejor boxeador del mundo.



Para empezar, su entrenador ha diseñado un plan de entrenamiento con pesas durante una serie de días. Cada día, Bruxo debe levantar ciertas pesas. Las pesas se repartirán entre los distintos días, de tal forma que cada una sea levantada algún día (solamente ese día), y en cada día se levante al menos una pesa.

Para hacer el entrenamiento más efectivo, el entrenador asegura que se deben repartir las pesas entre los distintos días de una forma particular. Para cada día, se halla la diferencia entre la de mayor y la de menor peso. Ahora, se ha de minimizar la suma de estas diferencias para todos los días.

Formalmente, sean $a_1, a_2, \ldots, a_n$ los pesos de cada una de las n pesas (los pesos pueden estar repetidos). El objetivo es distribuirlas entre los distintos días (obtener una partición) tal que cada pesa se utilice en un solo día, y en cada día se utilice al menos una pesa. El coste de una partición P es

$$c = \sum_{p_i \in P} \left(\max_{x \in p_i}(a) - \min_{x \in p_i}(a) \right)$$

donde p_i es el conjunto de pesas para el día i y $x \in p_i$ es una pesa de ese día. Se cumple que $1 \le i \le k$, siendo k el número de días.

Para cada k, con $1 \le k \le n$, encuentra el mínimo coste de las distribuciones de las pesas en k días.

Entrada

La entrada consiste en varios casos de prueba. La primera línea contiene un único entero t ($1 \le t \le 10^5$) — el número de casos de prueba. Cada caso de prueba está formado por dos líneas.

La primera línea contiene un único entero n ($1 \le n \le 10^5$) — el número de pesas.

La segunda línea contiene n enteros $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — los pesos de cada pesa.

Se garantiza que la suma de n sobre todos los casos de prueba no excederá 10^5.

Salida

Para cada caso de prueba, imprime n enteros $c_1, c_2, \ldots, c_n$ separados por espacios — siendo c_k el mínimo coste posible de distribuir las pesas en k días, para cada k ($1 \le k \le n$).

Entrada de ejemplo

```
3
1
5
2
3 7
3
5 2 3
```

Salida de ejemplo

```
0
4 0
3 1 0
```

Notas

En el primer caso de prueba:

- Para $k = 1$, tenemos $\{5\}$, con coste $5 - 5 = 0$.

En el segundo caso de prueba:

- Para $k = 1$, tenemos $\{3, 7\}$, con coste $7 - 3 = 4$.
- Para $k = 2$, tenemos $\{3\}, \{7\}$, con coste $(3 - 3) + (7 - 7) = 0$.

En el tercer caso de prueba:

- Para $k = 1$, tenemos $\{2, 5, 3\}$, con coste $5 - 2 = 3$.
- Para $k = 2$, hay tres posibles particiones: $(\{2\}, \{3, 5\})$, $(\{3\}, \{5, 2\})$ o $(\{5\}, \{3, 2\})$. Se puede comprobar que esta última tiene el menor coste, $(5 - 5) + (3 - 2) = 1$.
- Para $k = 3$, tenemos $\{3\}, \{2\}, \{5\}$, con coste $(3 - 3) + (2 - 2) + (5 - 5) = 0$.

Tiempo: 1 segundo

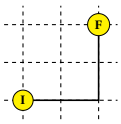
B
IKERobot

IKERobot es una marca de robots limpiadores *low-cost*. Como parte de la estrategia para reducir el precio de sus productos, IKERobot paga mal a sus ingenieros de software, por lo que tiene pocos ya que casi todos se han ido atraídos por los sueldos de las compañías de inteligencia artificial y las *startups* de *blockchain*. Como consecuencia, la programación del robot no está acabada y el comprador tiene que hacer parte de la misma. En particular, falta el código que guía al robot entre dos puntos de la vivienda.

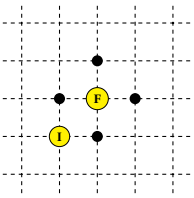
La representación que tiene el robot es la de un plano cuyos puntos están representados por coordenadas cartesianas (x, y) discretizadas. El robot sabe su posición y la del punto destino. También sabe la posición de los obstáculos (muebles, escaleras, paredes, mamparas, ...) representados mediante coordenadas de puntos que **no** puede atravesar. Se considera que el tamaño del robot es despreciable — es decir, podemos suponer que es un punto ideal 0-dimensional.

El robot solo puede moverse una unidad hacia adelante en paralelo a los ejes de coordenadas o rotar 90º hacia la izquierda o derecha. Si llega a un punto y avanza una unidad de espacio en la misma dirección en la que venía, ese desplazamiento cuesta una unidad de tiempo. Si al llegar a un punto decide rotar 90º, esta acción necesita cuatro unidades de tiempo. Si después hay un desplazamiento, este añade una unidad de tiempo. El código a realizar debe calcular el camino de **tiempo** mínimo, que puede ser distinto del de longitud mínima. El giro que el robot pueda hacer desde el punto inicial antes de desplazarse, si es necesario, no usa ninguna unidad de tiempo.

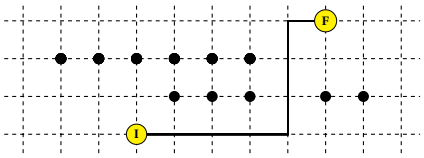
Los diagramas que siguen muestran tres posibles configuraciones y recorridos de tiempo mínimo. **I** denota la posición inicial del robot y **F** la final. Un disco marca una coordenada donde hay un obstáculo. Las líneas gruesas representan un camino de tiempo mínimo. La leyenda representa el coste de este camino en tiempo, si lo hay. Los diagramas se corresponden con los tres primeros casos de los ejemplos de entrada / salida.



Coste = 8



Coste = -1



Coste = 16

Entrada

En la primera línea, el número P de casos de prueba ($1 \leq P \leq 10$). Para cada caso de prueba, las coordenadas (x, y) (un par de enteros con $-50 \leq x, y \leq 50$) que representan los puntos de inicio I y de fin F. Seguidamente, el número de obstáculos, N ($0 \leq N \leq 100$). A continuación, N coordenadas (x_i, y_i) que representan puntos en los que el robot no puede estar. Las posiciones iniciales y finales del robot no coincidirán con ninguno de los puntos prohibidos para el robot. Aunque el valor de componentes de las coordenadas están acotados, el robot tiene la libertad de moverse fuera de esas cotas, es decir, por coordenadas con valores x o y mayores (respectivamente, menores) que 50 y -50.

Salida

Para cada caso de prueba, y en líneas distintas, el coste del camino entre los puntos de inicio y de final, si lo hay, o -1 si no existe el camino.

Entrada de ejemplo

```
4
0 0
2 2
0
1 1
2 2
4
1 2
2 1
3 2
2 3
3 0
8 3
11
1 2
2 2
3 2
4 2
5 2
6 2
4 1
5 1
6 1
8 1
9 1
-50 -50
50 50
2
49 50
50 49
```

Salida de ejemplo

```
8
-1
16
210
```

Tiempo: 2 segundos

C
Crimen en Villacepé

En la tranquila localidad de Villacepé ha ocurrido una tragedia: algún malhechor ha dejado hecho pedazos el ordenador del único turista que han tenido en los últimos 28 años. El agente A. C. toma declaración de los habitantes sobre quién podría ser el culpable. Con todos los datos, tiene ciertas sospechas, pero hay un problema: algunos habitantes son honestos y otros son mentirosos.

Un habitante honesto siempre dice la verdad, y uno mentiroso siempre miente. El agente está algo confundido, por lo que recoge nuevas declaraciones. Cada declaración es de una de estas dos formas:

- La persona i dice que la persona j es honesta.
- La persona i dice que la persona j es mentirosa.

Según los apuntes del agente, las personas no hacen declaraciones sobre ellas mismas y para cada par de personas hay, como mucho, una sola declaración. Es decir, si la persona i ha declarado sobre la persona j entonces la persona j **no** ha declarado sobre la persona i.

Dos personas están relacionadas si existe una declaración entre ellas **sin importar quién es el que declara**. Un grupo es un conjunto de personas donde para cualquier par de personas en el grupo se puede encontrar una sucesión de relaciones que las conecte.

Un ciclo es una sucesión de relaciones entre personas que empieza y acaba en la misma persona. Para simplificar, el agente apunta, como mucho, un ciclo por cada grupo de personas relacionadas.

Para empeorar más la situación, a la mañana siguiente el agente se encuentra con que el malechor ha roto sus apuntes, de modo que ahora solo se sabe que hubo una declaración de la persona i sobre la persona j, pero no sabe qué se declaró. El agente, ya derrotado, solo se le ocurre sacar su curiosidad matemática. Se pregunta cuántas configuraciones de posibles declaraciones (rellenar los huecos en las declaraciones con "honesta" o "mentirosa"), producen al menos una asignación consistente de honradez de todos los habitantes (rellenar los huecos en las personas con "honesta" o "mentirosa"). Necesita tu ayuda para obtener respuesta. Como esto puede ser un número muy grande, y eso le asusta un poco, obtén la respuesta módulo $10^9 + 7$. El módulo es el resto de la división entera.

Entrada

La entrada consiste en varios casos de prueba. La primera línea contiene un único entero t ($1 \le t \le 5 \cdot 10^4$) — el número de casos de prueba. Cada caso de prueba está formado por varias líneas.

La primera línea contiene dos enteros n ($2 \le n \le 2 \cdot 10^5$) y m ($1 \le m \le n \le 2 \cdot 10^5$) — el número de habitantes y el número de declaraciones, respectivamente.

A continuación aparecen m líneas, cada una con dos enteros $1 \le i, j \le n$, $i \ne j$ — las personas involucradas en la declaración. Se garantiza que los pares $\{i, j\}$ no se repiten.

Se garantiza que la suma de n sobre todos los casos de prueba no excederá $2 \cdot 10^5$.

Salida

Para cada caso de prueba, imprime el número (módulo $10^9 + 7$) de configuraciones válidas de las declaraciones.

Entrada de ejemplo

```
2
3 1
1 3
8 7
1 2
2 3
3 1
4 3
5 6
6 7
7 8
```

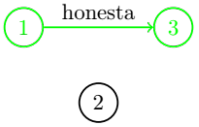
Salida de ejemplo

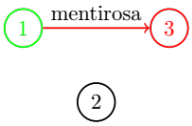
```
2
64
```

Notas

En el primer caso de prueba, pueden darse dos configuraciones:

- La persona 1 dice que la persona 3 es honesta. Para encontrar una asignación consistente vamos a suponer que la persona 1 es honesta, entonces dice la verdad, y por tanto, la persona 3 también es honesta. Ya que la persona 2 no interviene en ninguna declaración, da igual si es honesta o mentirosa. Hemos encontrado al menos una asignación válida para esta configuración de declaraciones. Por lo que la configuración nos sirve.
- La persona 1 dice que la persona 3 es mentirosa. Para encontrar una asignación consistente vamos a suponer que la persona 1 es honesta, entonces dice la verdad, y por tanto, la persona 3 es mentirosa. La persona 2 puede ser cualquier cosa, por ejemplo, honesta. También en este caso hay al menos una asignación válida para esta configuración.

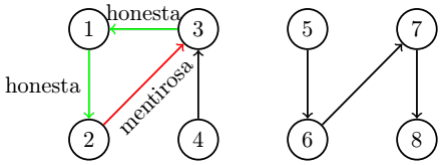




No hay más casos posibles de configuración de declaraciones, por lo que la respuesta al primer caso es 2.

En el segundo caso, un ejemplo de configuración de declaraciones que no tiene ninguna asignación válida de la honradez de las personas es alguna que incluya las declaraciones:

- La persona 1 dice que la persona 2 es honesta.
- La persona 2 dice que la persona 3 es mentirosa.
- La persona 3 dice que la persona 1 es honesta.



Tanto suponiendo que la persona 1 es honesta como suponiendo que es mentirosa, llegamos a una contradicción, luego esta asignación de aristas no es válida.

Tiempo: 4 segundos

D
El abeXORro

A *Xor*ge el abe*xor*ro le encanta el polen. Sabe por experiencias previas que en el jardín de Isidra hay muchas flores suculentas. Hoy vuelve a visitar el jardín pero, como ya ha comido suficiente, tiene otros planes. Le parece que el jardín de Isidra es mucho más bonito cuando todas las flores tienen la misma cantidad de polen, así que va a transportar polen entre flores hasta que todas tengan la misma cantidad.

A *Xor*ge, por algún motivo, le parece curiosa la operación XOR, así que transporta el polen de la siguiente forma: elige un par de flores, calcula el XOR de sus cantidades de polen y pone esa cantidad de polen en ambas flores, desperdiciando el resto.

Formalmente, sean las cantidades de polen en n flores: $a_1, a_2, \ldots, a_n$. En una operación, *Xor*ge puede elegir un i y un j tal que $1 \le i, j \le n$, $i \ne j$, y asignar $a_i := x$ y $a_j := x$, donde $x := a_i \oplus a_j$.

Dada una cantidad de polen k, *Xor*ge se pregunta si es posible obtener la misma cantidad k en todas las flores. Él es muy listo y ya ha resuelto el problema. Además, te garantiza que, si el problema tiene solución, entonces se puede llegar a ella con menos de $4n$ operaciones, así que te prohíbe hacer más. Recuerda que no es necesario minimizar el número de operaciones, sólo lograr que sean menos de $4n$. ¿Puedes encontrar tú también una solución?

Nota: La operación XOR de dos números enteros a y b, denotado en matemáticas como $a \oplus b$, es el número obtenido calculando el XOR bit a bit, también llamado XOR *bitwise*. El XOR de dos bits es 0 si son iguales, y 1 si son distintos. Por ejemplo, $5 \oplus 4 = 1$, ya que en binario se tiene $101_2 \oplus 100_2 = 001_2$.

Entrada

La entrada consiste en varios casos de prueba. La primera línea contiene un único entero t ($1 \le t \le 5 \cdot 10^4$) — el número de casos de prueba. Cada caso de prueba está formado por dos líneas.

La primera línea contiene dos enteros n ($4 \le n \le 2 \cdot 10^5$) y k ($0 \le k < 2^{31}$) — el número de flores y la cantidad requerida de polen por flor.

La segunda línea contiene n enteros $a_1, a_2, \ldots, a_n$ ($0 \le a_i < 2^{31}$) — las cantidades iniciales de polen de cada flor.

Se garantiza que la suma de n sobre todos los casos de prueba no excederá $2 \cdot 10^5$.

Salida

Para cada caso de prueba, imprime "`NO`", si no es posible llegar a tener todas las flores con el mismo polen tras cierto número de operaciones.

En caso contrario, imprime "`SI`" en una línea, en la siguiente indica el número de operaciones m ($0 \le m < 4n$) que vas a realizar. A continuación imprime m líneas, cada una con dos enteros $1 \le i, j \le n$ tal que $i \ne j$, las posiciones en las que se harán las operaciones, en orden. Recuerda que no es necesario encontrar el número mínimo de operaciones, pero debe ser estrictamente menor que $4n$. Dado que **la solución no es única**, cualquier solución que cumpla los requisitos será aceptada.

Dado que los archivos de salida que se van a producir pueden ser muy grandes, para los participantes que usen Java puede ser importante reducir el número de llamadas a `System.out.println`, la manera más sencilla de hacerlo es usar `StringBuilder` para acumular los datos de salida y hacer una única llamada a `System.out.println` al final de cada caso.

Entrada de ejemplo

```
6
5 1
0 1 1 0 1
4 4
6 4 0 2
4 0
1 0 0 1
4 0
1 0 0 1
4 2
0 1 1 0
5 76
48 38 26 121 2
```

Salida de ejemplo

```
SI
2
1 3
1 4
SI
2
1 4
3 2
SI
1
1 4
SI
2
3 2
4 1
NO
NO
```

Notas

En el primer caso de prueba se puede hacer una primera operación en las flores 1 y 3, quedando las cantidades de polen $[1,1,1,0,1]$, tras lo cual se puede hacer otra en las flores 1 y 4, quedando las cantidades $[1,1,1,1,1]$, que es lo que queríamos.

En el segundo caso de prueba se puede hacer una primera operación en las flores 1 y 4, quedando las cantidades de polen $[4,4,0,4]$, tras lo cual se puede hacer otra en las flores 2 y 3, quedando las cantidades $[4,4,4,4]$, que es lo que queríamos.

En los casos 3 y 4 se puede comprobar que cualquier solución válida es aceptada.

En los casos 5 y 6 se puede demostrar que no es posible hacer que todas las flores tengan la cantidad de polen requerida.

Tiempo: 1 segundo

E
Obras de ingeniería

Los habitantes de la localidad de Villacepé saben que recibir turistas es casi imposible. En un intento de embellecerla, el alcalde decide hacer ciertas reformas en los edificios. La calzada en la avenida principal tiene un único sentido, y es también la vía de entrada a Villacepé. El alcalde piensa que los turistas se asombrarán más si, a medida que recorren la avenida al entrar, los sucesivos edificios nunca reducen su número de plantas respecto a los anteriores, es decir, le gustaría que los edificios formaran una fila con plantas no decrecientes.

Para conseguirlo, el alcalde solicitará la ayuda de los vecinos. Estos son un poco extravagantes, y solo aceptarán tareas dadas en forma de operaciones muy específicas. En una operación, el alcalde les puede indicar un grupo de edificios consecutivos y un número de plantas h, y ellos destruirán o construirán plantas hasta que todos los edificios del grupo tengan h plantas. El coste de una operación es la suma de las diferencias absolutas de plantas entre las anteriores y las nuevas de cada edificio.

Formalmente, sean el número de plantas de los edificios en orden $p_1, p_2, \ldots, p_n$. En una operación, el alcalde puede elegir dos enteros $1 \le a \le b \le n$ y un entero $1 \le h$, y cambiar $p_i := h$ para cada $a \le i \le b$. El coste asociado a esta operación es $\sum_{i=a}^{b} |p_i - h|$.

El alcalde tiene buen presupuesto de lo que consigue ganando concursos de programación competitiva, así que tendrá suficientes fondos para llevar a cabo sus curiosos planes, aún así le gusta gastarse lo justo (quizá no gana tanto como dice), así que quiere saber cuál es el coste mínimo de las operaciones necesarias para que se cumpla su plan de reformar los edificios de manera que el número de plantas no decrezca. Necesita tu ayuda porque... Bueno, parece que no puede resolver este problema por sí mismo, y ha decidido rendirse hasta que publiquen la solución.

Entrada

La entrada consiste en varios casos de prueba. La primera línea contiene un único entero t ($1 \le t \le 2000$) — el número de casos de prueba. Cada caso de prueba está formado por dos líneas.

La primera línea contiene un único entero n ($1 \le n \le 5000$) — el número de edificios.

La segunda línea contiene n enteros $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le 5000$) — el número de plantas de los edificios.

Se garantiza que la suma de n sobre todos los casos de prueba no excederá 5000.

Salida

Para cada caso de prueba, imprime un único entero — el mínimo coste que se requiere para reformar los edificios de tal forma que el número de plantas sea no decreciente. Recuerda que no es necesario minimizar el número de operaciones, solo el coste.

Entrada de ejemplo

```
3
1
1
2
3 1
3
4 2 4
```

Salida de ejemplo

```
0
2
2
```

Notas

En el primer caso de prueba hay un único edificio, por lo que no hay que compararlo con ningún otro, y el coste es 0.

En el segundo caso de prueba, el coste mínimo es 2, con el cual se pueden obtener por ejemplo los edificios con alturas $2, 2$, realizando una operación con $a = 1$, $b = 2$, $h = 2$. Esta no es la única forma, pudiendo también obtener las alturas $3, 3$ con $a = 2$, $b = 2$, $h = 3$ o las alturas $1, 1$ con $a = 1$, $b = 1$, $h = 1$ también con coste total 2.

En el tercer caso de prueba, el coste mínimo es 2, con el cual se pueden obtener por ejemplo los edificios con alturas $3, 3, 4$, realizando una operación con $a = 1$, $b = 2$, $h = 3$.

Tiempo: 2 segundos

F
Fechas de entrenamiento

Un equipo de programación se está preparando para un próximo concurso de programación y han elegido a Leo, un ex-participante y entrenador de programación experimentado, para que los entrene. Leo, que es muy organizado, ha proporcionado al equipo una lista de fechas en las que les entregará problemas de programación y la cantidad de problemas que les propondrá en cada fecha.

Sin embargo, hay una restricción importante: el equipo tiene un límite de tiempo para trabajar juntos. Debido a las obligaciones laborales y personales de los miembros del equipo, solo pueden reunirse un máximo de K días.

En cada reunión, el equipo resolverá todos los problemas que Leo les ha propuesto hasta esa fecha (incluyendo los problemas propuestos para el mismo día de la reunión) que aún no han resuelto desde la última reunión.

El objetivo del equipo es resolver todos los problemas de Leo y minimizar la suma de la cantidad total de días que transcurren desde la fecha en que Leo les propone un problema hasta la fecha en que el equipo lo resuelve.

Por ejemplo, supongamos que Leo les entrega una lista de problemas para las próximas tres fechas donde en la primera fecha les propone un problema, en la segunda fecha dos problemas y en la tercera fecha tres problemas. Si el equipo se puede reunir en una sola fecha (K=1), lo óptimo es que lo hagan en la tercera fecha. De esta forma, para el problema de la primera fecha pasan 2 días entre que Leo se los propone y el equipo lo resuelve, para los dos problemas de la segunda fecha pasa 1 día entre la fecha que Leo se los propone y el equipo lo resuelve y los tres problemas de la tercera fecha los resuelven en el mismo día que Leo se los propone. Por lo tanto, la cantidad total de días que transcurren desde la fecha en que se propone un problema hasta la fecha en que se resuelve es de $2 \cdot 1 + 1 \cdot 2 + 0 \cdot 3 = 4$.

¿Puedes escribir un programa para ayudar al equipo a planificar sus reuniones y alcanzar su objetivo de resolver todos los problemas de Leo minimizando la suma de la cantidad total de días que transcurren desde la fecha en que se propone un problema hasta la fecha en que lo resuelven?

Entrada

La primera línea contiene dos enteros n ($1 \le n \le 10^3$) y k ($1 \le k \le n$) — el número de días que Leo va a proponer problemas y la cantidad de días que el equipo se puede juntar, respectivamente.

A continuación aparecen n líneas, cada una con dos enteros d y p ($1 \le d, p \le 10^5$), representando que Leo propone p problemas en el día d. Se garantiza que los valores de d están en orden estrictamente creciente.

Salida

Para cada caso de prueba, se imprimirá la mínima suma de la cantidad total de días que transcurren desde la fecha en que Leo propone un problema hasta la fecha en que el equipo lo resuelve.

Entrada de ejemplo

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

Salida de ejemplo

```
4
```

Entrada de ejemplo

```
7 3
1 8
3 13
5 14
6 9
8 12
9 5
10 11
```

Salida de ejemplo

```
59
```

Notas

En el primer caso de prueba, para poder pensar todos los problemas deben juntarse el día 3, lo que tiene un costo de $(3-1)\cdot 1+(3-2)\cdot 2+(3-3)\cdot 3=4$.

En el segundo caso, se puede ver que el óptimo se obtiene si se juntan los días 3, 6 y 10, lo que da un costo de $(3-1)\cdot 8+(3-3)\cdot 13+(6-5)\cdot 14+(6-6)\cdot 9+(10-8)\cdot 12+(10-9)\cdot 5+(10-10)\cdot 11=59$.

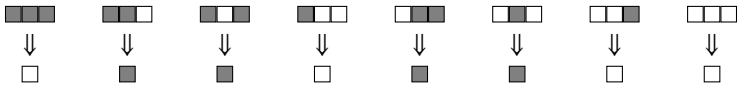
Tiempo: 2 segundos

G
El jardín del Edén

Un autómata celular lineal elemental consiste en una serie de C celdas (células) adyacentes que pueden estar vivas (■) o muertas (□), más una regla de evolución que, dado un estado del autómata, determina el siguiente estado. Por ejemplo, para un autómata con $C = 8$ células una posible evolución sería

Asignaremos un número a cada estado interpretándolo como una cifra binaria donde una célula viva se corresponde con el `1` y una muerta con el `0`. Los estados anteriores serían `10110101`, `11111111` y `10000001` o, en decimal, 181, 255 y 129.

Una regla de evolución determina el siguiente estado de una célula usando su estado actual y el de sus dos vecinas. Supondremos (sin representarlas) que a ambos lados de la configuración hay células muertas que nunca cambian. Por ejemplo, hacer que una célula esté viva en el siguiente instante de tiempo si y solo está ya viva y sin vecinas o si, entre ella y sus vecinas hay exactamente dos células vivas daría la siguiente regla:



Una regla se puede numerar interpretando cada grupo de tres células adyacentes como un número binario b y el siguiente estado asociado a esa configuración como el valor del bit que ocupa la posición b-ésima en un número de 8 bits, el cual identifica a la regla. La regla anterior tendría el número binario `01101100` o, en decimal, el 108. La evolución mostrada al principio del enunciado usa esta regla.

Para algunas reglas existen configuraciones que **no** pueden provenir de la evolución de ninguna otra configuración **diferente** y se denominan *jardines del Edén*. La primera configuración del ejemplo de evolución de este enunciado es un jardín del Edén para la regla que hemos mostrado. El problema a resolver es determinar si una configuración dada es o no un jardín del Edén para una regla también dada.

Entrada

En una línea, el número de casos de prueba, N ($1 \leq N \leq 5000$) Para cada caso de prueba, y en una linea cada caso, el número R de regla ($0 \leq R \leq 255$), el número C de células de la configuración ($1 \leq C \leq 30$) y el número A de la configuración ($0 \leq A < 2^C$).

Salida

Para caso de prueba se debe escribir, en líneas separadas, `SI` si la configuración es un jardín del Edén y `NO` en caso contrario.

Entrada de ejemplo

```
5
108 8 181
90 10 111
204 16 43690
2 16 0
108 30 346521633
```

Salida de ejemplo

```
SI
NO
SI
NO
NO
```

Tiempo: 2 segundos

H
El máximo de diversión

En muchas ocasiones los niños pequeños nos hacen replantearnos algunos hechos que consideramos indiscutibles. Cualquiera que haya paseado con *gente menuda* sabe perfectamente que los *peques* aplican sus propias reglas para calcular la distancia entre dos puntos. Mientras que un adulto cuando tiene que ir del punto A al punto B normalmente busca el camino más fácil, los niños buscan ¡El que gasta más energía! Bueno, quizás pueda ser el camino que les parece más divertido, pero a ojos de los adultos sin duda es el más difícil y el que más energía consume.

Naturalmente, esta peculiar visión del mundo de la *gente menuda* tiene algunas ventajas si sabes aprovecharlas. Una de ellas son los paseos a la carrera: si tus *peques* empiezan a protestar de que el paseo es muy largo lo mejor que puedes hacer es proponerles una carrera, en ese momento observarás verdaderamente sorprendido cómo el cansancio desaparece y los pequeñajos se lanzan a correr como si nada.

El instituto para el estudio de la niñez está investigando este tipo de comportamiento. Hasta ahora ha encontrado una fórmula que permite estimar cuál es el grado de diversión (de energía gastada) que los niños asignan a ir de un punto a otro. Además, han diseñado una serie de circuitos caracterizados por una sucesión de puntos que se deben recorrer en orden y sin saltarse ningún punto. Lo que quieren saber es el valor de la diversión máxima cuando van de un punto a otro en esos circuitos.

Los circuitos están formado por n puntos y se definen mediante las coordenadas de los puntos (en 2 dimensiones). Para cualquier par de puntos consecutivos en la ruta: (x_i, y_i) e (x_{i+1}, y_{i+1}), la diversión estimada para ir del punto P_i a P_{i+1}, denotada $D_{i->i+1}$, viene dada por

$$D_{i->i+1} = 3 \cdot |x_{i+1} - x_i| + 2 \cdot (y_{i+1} - y_i)$$

Es decir, 3 veces el valor absoluto de la diferencia de las coordenadas x más el doble de la diferencia de las coordenadas y. El máximo que se desea calcular es entonces:

$$maxD = \max_{1 \le i < n} D_{i->i+1}$$

Entrada

La entrada consiste en la descripción de varios circuitos. Cada circuito es un caso de prueba. La primera línea contiene un único entero que indica el número de circuitos. Las descripciones de cada circuito están formadas por dos líneas.

La primera línea contiene un único entero n ($2 \le n \le 10^3$) que indica el número de puntos en 2D que forman el circuito.

La segunda línea contiene $2n$ enteros que indican las coordenadas de los puntos, es decir, $x_1, y_1, x_2, y_2 \ldots, x_n, y_n$, donde ($-10^2 \le x_i, y_i \le 10^2$).

El número máximo de casos de prueba es 200, se debe asumir que todos ellos pueden tener 1000 puntos.

Salida

Para cada caso de prueba, imprime la diversión máxima.

Entrada de ejemplo

```
3
2
5 8 3 2
3
3 7 5 8 3 2
3
5 8 3 2 -1 -5
```

Salida de ejemplo

```
-6
8
-2
```

Tiempo: 2 segundos

I
Razonamiento Numérico

En la Facultad de Psicología están muy interesados en rediseñar sus test psicotécnicos, especialmente los relacionados con razonamiento numérico. Como en general sus estudiantes son más aficionados a las letras que a los números, han convocado un concurso para que estudiantes de informática propongan problemas de números con leyes muy difíciles y así tener desafíos verdaderamente exigentes.

Tu equipo participa en el concurso y naturalmente quiere acabar en primera posición, así que os habéis puesto manos a la obra y habéis diseñado una fórmula verdaderamente enrevesada que seguro está a la altura de las circunstancias. El problema es que ahora hay que generar ejemplos que apliquen la fórmula y para ello habéis pensado que lo más seguro es hacer un programa que permita generar esos ejemplos de manera automática no vaya a ser que, al hacerlo de cabeza, se pueda introducir algún error.

Vuestra formula se basa en aplicar una transformación a un número muy largo para obtener otro número aproximadamente con el doble de cifras. Para ello se deben tomar las cifras de dos en dos desde la cifra más significativa (el número original tiene número de cifras pares). Llamaremos x e y a la primera y segunda cifra de cada par de cifras, se sabe que el número solamente contiene cifras distintas de cero, es decir $1 \leq x, y \leq 9$. A partir de x e y, se calcula un número r dado por:

$$r_1 = \begin{cases} y^6, & \text{si } x = 1 \\ (y^6)! \ \textbf{mod}\ (10^9 + 7), & \text{en caso contrario} \end{cases}$$

$$r_2 = \begin{cases} x^6, & \text{si } y = 1 \\ (x^6)! \ \textbf{mod}\ (10^9 + 7), & \text{en caso contrario} \end{cases}$$

$$r = (47 \cdot r_1 + r_2) \ \textbf{mod}\ 9999 + 1$$

Siendo **mod** el operador módulo, es decir, resto de la división entera.

Finalmente, para calcular el resultado se utilizan los sucesivos valores de r como las cifras del número resultante también tomadas a partir de la cifra más significativa.

Entrada

La entrada consiste en varios casos de prueba. Cada caso de prueba es un entero a con un número par de cifras todas ellas distintas de cero tal que $1 \leq a < 10^{1022}$. La entrada terminará con un cero que no genera ninguna salida.

El número máximo de casos de prueba es 2000, se debe asumir que todos ellos pueden tener un número, a, con el máximo de número de cifras (1022).

Salida

Para cada caso de prueba, se imprime el número resultado de hacer la transformación indicada.

Entrada de ejemplo

```
12
32
1232
2112
12322112
0
```

Salida de ejemplo

```
3010
4935
30104935
1123010
3010493511230l0
```

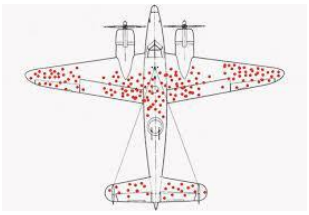
Notas

- Para el primer caso de prueba, 12, el número solo tiene dos cifras, así pues $x = 1, y = 2$ y aplicando las expresiones obtenemos: $r1 = 64, r2 = 1$ y finalmente $r = 3010$.
- En el segundo caso de prueba, 32, tenemos que $x = 3, y = 2$, aplicando de nuevo las expresiones obtenemos: $r1 = 331333826, r2 = 29224765$ y $r = 4935$. Para calcular $r1$ es necesario calcular $(2^6)!$ **mod** $(10^9 + 7)$; 2^6 es 64, el factorial de 64 es muy grande, de hecho es aproximadamente $1,2688 \cdot 10^{89}$, pero al aplicar el módulo se queda en $r1 = 331333826$. El cálculo de $r2$ es análogo, es necesario calcular el factorial de 729 cuyo resultado aproximado es $1,4394 \cdot 10^{1772}$, de nuevo un número extraordinariamente grande, pero al aplicar el modulo obtenemos $r2 = 29224765$.
- En el tercer caso de prueba primero se debe descomponer el número 1232 en pares de cifras, se obtienen los pares 12 y 32 y se procede como en los dos casos anteriores. Para obtener el resultado final se compone el número a partir de los resultados de cada par de cifras.
- Se procede de forma similar para el resto de casos.

Tiempo: 1 segundo

J
Aviones en Guerra

Durante la Segunda Guerra Mundial, el gobierno de Estados Unidos se planteó mejorar el blindaje de sus aviones de combate añadiendo planchas de metal reforzando el avión. Añadir planchas por todo el avión aumentaba el peso del mismo, reduciendo su velocidad y aumentando el de combustible. Así que decidieron reforzar solo las partes claves del avión.



Para determinar qué partes eran las más importantes, analizaron los agujeros de bala de los aviones que regresaban, y reforzaron las partes con más agujeros.

Los investigadores se dieron cuenta más tarde que el planteamiento estaba mal: si un avión había regresado, el disparo no había impactado en una zona clave. ¡Estaban reforzando las zonas que no necesitaban refuerzo! La solución propuesta fue proteger las partes que no habían recibido disparos.

En estadística, este fenómeno se denomina sesgo de supervivencia, que, como hemos visto, consiste en concentrarse en las personas o elementos que superaron un proceso de selección pasando por alto a aquellas que no lo hicieron.

En la Segunda Guerra Mundial ya se usaban algunos ordenadores, por ejemplo, para romper cifrados. Sin embargo, su capacidad de analizar aviones era relativamente limitada. El objetivo es, dada una serie de imágenes de aviones y listados de coordenadas indicando dónde recibieron los impactos, realizar un escáner que indique cuál es la zona que necesita ser reforzada.

Entrada

La entrada consiste en un solo caso de prueba. La primera línea contiene dos enteros M y N ($5 \leq M, N \leq 100$), el ancho y el alto en píxeles de la imagen tomada. Seguidamente, vendrán los $N \cdot M$ píxeles de la imagen en N líneas con M píxeles cada una.

Cada píxel será un "." (punto), que representa parte del fondo (no es parte del avión) o un número del 0 al 9, donde cada número representa una parte diferente del avión. Todas las fotos tendrán al menos un píxel como parte del avión.

A continuación se mostrará un número D ($1 \leq D \leq 2000$), que denotará el número de impactos registrados. Cada impacto se representa en una línea con dos enteros (y_i, x_i), la fila y la columna del píxel en la imagen respectivamente ($0 \leq y_i < N$, $0 \leq x_i < M$). Se garantiza que los impactos siempre son sobre una zona del avión, es decir, no habrá ninguno en un píxel que represente el paisaje. Además, entre los impactos no se tienen registros repetidos, es decir, en cada posición solo se dará como mucho un impacto.

Salida

Para cada caso de prueba, se imprimirá la zona a reforzar, es decir, aquella que menos disparos haya recibido. Si hay más de una solución, se devolverá la zona de menor tamaño, ya que sabemos que el peso del avión es muy importante para su correcto funcionamiento. En caso de seguir habiendo más de una solución, se devolverá la zona con el menor identificador.

Entrada de ejemplo

```
50 10
.....11111111
.....100000001
.....100000001.
.....1111111111111111111111111111111100000001..
...10008000000008000800080080008000800000000001...
.10000000000100000000010000000000000000001.....
100000000000010000000001000000000000000001.......
.11111111111111111111111111111111.............
.................1222222221.....................
.................111111111.....................
5
5 20
6 17
6 0
4 23
5 38
```

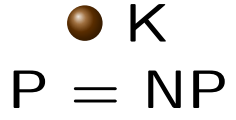
Salida de ejemplo

```
8
```

Notas

De los 5 disparos, 4 de ellos dan en la zona “0” y el tercer disparo da en la zona “1” del avión (recuerda que las coordenadas empiezan en 0). Por tanto, hay empate con menor número de disparos (0) entre las zonas presentes “2” y “8”. Atendiendo ahora al tamaño de cada zona, vemos que la zona “8” es la más pequeña con 7 píxeles, por lo que ya no hay empate y la zona “8” es la respuesta.

Tiempo: 1 segundo



Lo que se conoce como “problema P vs NP” es algo que trae de cabeza a muchos científicos. Informalmente, se trata de intentar demostrar si se puede calcular en tiempo polinomial (P) la solución a cualquier problema si esta solución se puede verificar en tiempo polinomial (NP).

Sin embargo, a pesar de que aún nadie ha podido demostrarlo, nosotros sabemos que P = NP en una serie de casos:

- P = NP siempre que P=0, para cualquier otro valor de N.
- P = NP siempre que N=1, para cualquier otro valor de P.

Además de estos casos, en nuestro ordenador es posible que veamos algunos más, dependiendo de qué tipo de datos utilicemos para almacenar los valores y para hacer las operaciones. En concreto, si el ordenador utiliza números enteros sin signo de 8 bits (con valores entre el 0 y el 255), al multiplicar los valores P=128 y N=3 nos daría el número 384, pero como no se puede representar con solamente 8 bits, hay desbordamiento y el ordenador mostraría como resultado 128, que es igual a P.

En este problema, tienes que descubrir cuántos valores de P cumplen la ecuación P = NP en un ordenador con desbordamiento, si todas las operaciones se hacen con enteros sin signo de 32 bits (*uint32*).

Entrada

La entrada consiste en varios casos de prueba. La primera línea contiene un único entero t ($0 \leq t \leq 10^5$), que indica el número de casos de prueba.

Las siguientes t líneas tienen valores de N ($0 \leq N < 2^{32}$).

Salida

Para cada caso de prueba, se imprimirá el número de valores distintos de P que hagan que se cumpla la ecuación.

Entrada de ejemplo

```
4
2
3
4
5
```

Salida de ejemplo

```
1
2
1
4
```