XXX Olimpiada Informática Española

OIE 2026 — Final Día 2

12 de abril de 2026

Organizan y financian

Patrocinan

Listado de problemas

Autores y revisores de los problemas: María Lucía Aparicio, Max Balsells, Izan Beltran, Hugo Domínguez, Blanca Huergo, Roger Lidón, Huize Mao, Darío Martínez, Edgar Moreno, Daniel Nieto, Pablo Sáez, Manuel Torres, Anier Velasco, Alberto Verdejo, Alejandro Vivero.

La Picasso de Mnieto

COCHE

Dani ha perdido el autobús y no quiere llegar tarde a casa. Para ahorrar tiempo, piensa en llamar a su hermano Marco y pedirle que pase a recogerlo en su coche.

La ciudad de Valladolid se puede modelar como un grafo no dirigido con n nodos numerados del 1 al n y m aristas.

- Marco comienza en el nodo 1 y Dani en el nodo 2.
- Marco conduce y tarda 1 segundo en cruzar una arista.
- Dani camina y tarda 2 segundos en cruzar una arista.
- Tanto Dani como Marco pueden decidir parar un tiempo en un nodo antes de volver a ponerse en marcha.
- Dani puede acordar encontrarse con Marco en cualquier nodo, subirse a su coche instantáneamente y, desde ese momento, continuar moviéndose con la velocidad de Marco.

Para cada nodo i del 1 al n, determina el tiempo mínimo que tarda Dani en ir del nodo 2 al nodo i si Marco y él se mueven de forma óptima.

Entrada y salida

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

Para cada caso:

- La primera línea contiene dos enteros n y m: el número de nodos y aristas del grafo.
- Las siguientes m líneas contienen dos enteros u y v cada una, indicando que existe una arista no dirigida entre los nodos u y v.

Se garantiza que el grafo es conexo y no contiene aristas múltiples (aristas que conecten el mismo par de nodos) ni bucles (aristas que conecten un nodo consigo mismo).

Para cada caso, imprime una línea con n enteros. El i-ésimo entero debe ser el tiempo mínimo necesario para que Dani llegue del nodo 2 al nodo i.

Restricciones

- $1 \leq T \leq 100$
- $2 \leq n \leq 10^5$
- $n - 1 \leq m \leq \min\left(\frac{n \cdot (n-1)}{2}, 2 \cdot 10^5\right)$
- $1 \leq u, v \leq n$
- La suma de n sobre todos los casos no supera 10^5.
- La suma de m sobre todos los casos no supera $2 \cdot 10^5$.

Subtareas

1. (16 puntos) Los nodos 1 y 2 son adyacentes.
2. (25 puntos) El grafo es un árbol (no contiene ciclos).
3. (19 puntos) La suma de n sobre todos los casos no supera 1000, y la de m no supera 2000.
4. (40 puntos) Sin restricciones adicionales.

Ejemplo

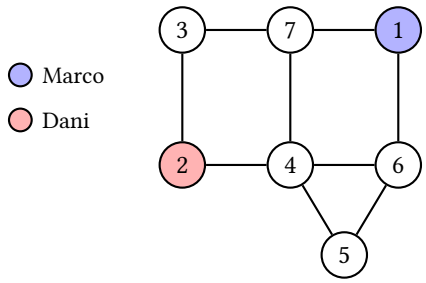
Entrada

```
3
6 9
2 1
3 6
4 3
6 5
4 1
4 2
3 2
3 1
5 4
7 9
7 1
7 3
6 1
6 4
4 2
7 4
6 5
5 4
3 2
8 7
4 1
6 3
4 3
7 3
5 2
6 5
8 7
```

Salida

```
2 0 2 2 3 3
4 0 2 2 3 3 3
7 0 5 6 2 4 6 7
```

En el **segundo caso**, el grafo se puede modelar de la siguiente manera:



Analicemos cómo llegar lo más rápido posible a algunos nodos:

- **Para alcanzar el nodo 1:** Dani podría encontrarse con Marco en los nodos 3 o 4 tras 2 segundos. Después de 2 segundos más, llegarían en coche al nodo 1, sumando un total de 4 segundos.
- **Para alcanzar el nodo 5:** Dani podría encontrarse con Marco en el nodo 4 tras 2 segundos (Dani tarda 2 segundos en recorrer la arista $2 \to 4$, y Marco tarda 2 segundos en recorrer las aristas $1 \to 6$ y $6 \to 4$). Un segundo después, llegarían en coche al nodo 5, completando un total de 3 segundos.
- **Para alcanzar el nodo 7:** Dani recorre la arista $2 \to 3$ en 2 segundos. Simultáneamente, Marco recorre $1 \to 7$ y $7 \to 3$ en 2 segundos (1 segundo por arista). Una vez se encuentran en el nodo 3,

Dani sube al coche y en 1 segundo vuelven al nodo 7 mediante la arista $3 \to 7$. El tiempo total es de 3 segundos.

- **Para alcanzar el nodo 4:** Dani puede ir caminando directamente por la arista $2 \to 4$ y llegar en 2 segundos. En este caso, no existe ninguna forma de llegar más rápido, ni siquiera coordinándose para encontrarse con Marco en otro punto.

Pagos al final del viaje

PAGOS

Es la final de la OIE y, un año más, Jacobo ha pagado más de lo que le tocaba y todo el mundo debe hacerle transferencias para saldar las cuentas. Pero claro, esto es lioso, ya que Blanca también ha adelantado el dinero de las comidas, Manu ha comprado los refrescos, etc.

Darío sugiere usar Splitwise, una aplicación en la que los n voluntarios apuntan cuánto han pagado cada vez que hacen una compra. Al final del viaje, esta aplicación suma cuánto ha pagado cada uno y, usando un algoritmo inteligente, organiza las transferencias para que cada voluntario acabe pagando lo mismo. Además, con este algoritmo, saldar las cuentas requiere como máximo $n - 1$ transferencias. ¿Sabrías diseñar un algoritmo semejante?

Entrada y salida

Recibirás una línea en la entrada con un entero T, el número de casos que has de procesar. Cada caso consistirá en:

- Una línea con dos enteros n y m: el número de voluntarios y el número de compras hechas en el viaje, respectivamente.
- Una línea con n cadenas de caracteres: los nombres de los voluntarios.
- m líneas, cada una con dos valores: s_i, el nombre del voluntario que hizo el i-ésimo pago; y c_i, un entero que representa la cuantía de ese pago en céntimos.

Por cada caso de prueba, imprime una línea con un entero p: el número de transferencias con el que se podrían saldar todas las cuentas. p debe ser menor o igual que $n-1$. Después, imprime p líneas (una por transferencia, en cualquier orden), donde describas (1) quién hace la transferencia, (2) quién la recibe y (3) de qué valor es la transferencia (un valor estrictamente positivo). Observa que **no** es necesario minimizar el número de transferencias, pero este debe ser como máximo $n - 1$. Cualquier solución que cumpla estas restricciones será válida.

Restricciones

- $1 \leq T \leq 100$
- $2 \leq n \leq 50.000$
- $1 \leq m \leq 50.000$
- $1 \leq c_i \leq 10.000$
- En cada caso de prueba, la suma de las c_i es múltiplo de n. Es decir, se garantiza que hay una solución.
- Los nombres de los voluntarios tendrán entre 1 y 20 caracteres, todos ellos en el alfabeto inglés. En cada caso de prueba, todos los nombres serán distintos entre sí.
- La suma de las n de todos los casos de prueba en un mismo archivo no superará 50.000.
- La suma de las m de todos los casos de prueba en un mismo archivo no superará 50.000.

Subtareas

1. (10 puntos) Se necesitará exactamente una transferencia para saldar las cuentas.
2. (15 puntos) Como máximo, dos transferencias serán suficientes para saldar las cuentas.

3. (19 puntos) $n \leq 1000$ y $m \leq 1000$.
4. (21 puntos) Cada voluntario ha hecho, como mucho, un pago.
5. (35 puntos) Sin restricciones adicionales.

Ejemplo

Entrada

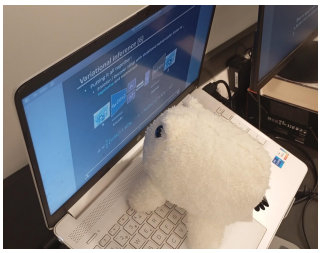
```
3
4 1
Jacobo Marco Pedro Joan
Jacobo 400
2 3
Jacobo Marco
Jacobo 1
Jacobo 2
Marco 3
3 3
dario manu alberto
dario 4
manu 5
alberto 6
```

Salida

```
3
Joan Jacobo 100
Pedro Jacobo 100
Marco Jacobo 100
0
1
dario alberto 1
```

Baba has exam

EXAMEN



Baba tiene un examen de informática pronto. En clase han dado n temas y el examen tendrá n preguntas, una de cada tema. Tras hacer los cálculos, ha concluido que necesita acertar al menos k de ellas para que la evaluación final le salga aprobada.

Baba actualmente tiene un nivel de w_i en el i-ésimo tema. Este w_i es un entero (posiblemente negativo, si va muy perdido) que representa cuánto se ha enterado de lo que ha explicado el profesor. Quiere estudiar unas cuantas horas más (posiblemente un número decimal de horas), que se puede repartir como quiera entre los diferentes temas. Si estudia el i-ésimo tema durante h_i horas (h_i puede ser decimal), su nivel de ese tema aumentará en h_i y pasará a ser $w_i + h_i$.

El profesor ha prometido que pondrá un examen de dificultad D, lo cual significa que pondrá preguntas con dificultades que sean no negativas y sumen D. Es decir, si llamamos d_i a la dificultad de la pregunta del i-ésimo tema (que puede ser decimal), el profesor pondrá un examen tal que $d_1 + \ldots + d_n = D$ y $d_i \geq 0$ para todo i. Baba acertará la i-ésima pregunta si y solo si tiene nivel al menos d_i en ese tema, es decir, si $w_i + h_i \geq d_i$.

Ayúdale a saber cuál es el mínimo número de horas de estudio que necesita para asegurarse acertar k preguntas, sin importar qué examen ponga el profesor.

Entrada y salida

La primera línea contiene un entero T, el número de casos.

Siguen T casos, cada uno ocupando 2 líneas:

- La primera línea contiene 3 enteros: n, el número de temas; k, el número de preguntas que necesita acertar; y D, la dificultad del examen.
- La segunda línea contiene n enteros: $w_1, \ldots, w_n$, el nivel de Baba en cada tema.

Por cada caso, escribe una línea con un solo número decimal, la cantidad mínima de horas de estudio que Baba necesita para garantizar acertar k preguntas. Tu respuesta será considerada correcta si el mínimo del error absoluto y relativo[1] entre tu respuesta y la de los jueces es como mucho 10^{-4}.

Para asegurar que tu solución sea evaluada correctamente, **recomendamos imprimir tu resultado siempre con 6 cifras decimales**. Suponiendo que tienes una variable decimal `x`, puedes imprimirla con 6 cifras decimales escribiendo:

- `cout << fixed << setprecision(6) << x << "\n";` en C++.
- `print(f"{x:.6f}")` en Python.
- `System.out.printf("%.6f\n", x);` en Java.

Restricciones

- $1 \leq T \leq 4 \cdot 10^4$
- $2 \leq n \leq 2 \cdot 10^5$
- $1 \leq k \leq n$

[1] Si tu respuesta es r y la de los jueces es R, el mínimo de los errores absoluto y relativo es $\frac{|R-r|}{\max(1,R)}$.

- $1 \leq D \leq 5 \cdot 10^3$
- $-5 \cdot 10^3 \leq w_i \leq 5 \cdot 10^3$
- La suma de n sobre todos los casos es como mucho $2 \cdot 10^5$.

Subtareas

1. (10 puntos) $k = 1$.
2. (8 puntos) $k = n$.
3. (17 puntos) $k = n - 1$.
4. (20 puntos) $w_i \geq 0$ para todo $1 \leq i \leq n$.
5. (15 puntos) $w_i \leq 0$ para todo $1 \leq i \leq n$.
6. (30 puntos) Sin restricciones adicionales.

Ejemplo

Entrada

```
3
2 2 1
2 1
7 4 5
3 5 -1 3 -1 3 -2
3 2 3
0 0 0
```

Salida

```
0.000000
3.000000
4.500000
```

En el **primer caso**, el nivel inicial de Baba en todos los temas ya es igual o superior a la dificultad total de cualquier examen ($D = 1$). Por lo tanto, Baba no necesita invertir ninguna hora extra de estudio y el coste es 0.

En el **segundo caso**, una de las distribuciones óptimas es estudiar el tercer tema durante 3 horas, y no estudiar ninguno más. De esta manera, su array de niveles después de estudiar será $3, 5, 2, 3, -1, 3, -2$, y se puede demostrar que sirve para acertar 4 preguntas en cualquier examen con dificultad 5.

Por ejemplo, si el profesor pone un examen con dificultades $0, 1, 2, 2, 0, 0, 0$, Baba acertará las preguntas 1, 2, 3, 4 y 6. Si pone un examen con dificultades $0, 1, 3, 1, 0, 0, 0$, acertará las preguntas 1, 2, 4 y 6.

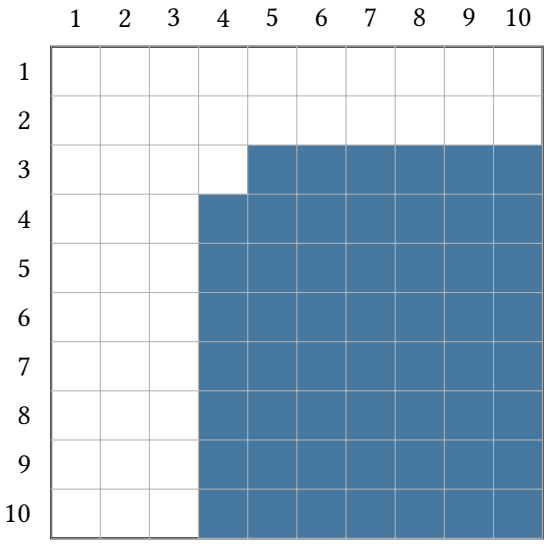
En el **tercer caso**, Baba puede estudiar cada uno de los temas durante 1.5 horas. De esta manera, se garantiza acertar al menos 2 preguntas para cualquier examen de dificultad 3.

Ataque par

TABLERO

Este problema es *output-only*, lo que significa que no hay entrada.

En un tablero de ajedrez de dimensiones 100×100 se quieren poner piezas en algunas de las casillas. Las piezas de este tablero son algo especiales. Una pieza situada en la casilla (i, j) ataca a todas las casillas (x, y) que cumplan $x \geq i$ y $y \geq j$, excepto a su propia casilla. Por ejemplo, en este tablero 10×10, una pieza en la casilla $i = 3, j = 4$ ataca a las casillas coloreadas:



Decimos que una casilla es **buena** si, al colocar una pieza en ella, esta atacaría a un número par de piezas (es decir, hay un número par de piezas en las casillas que domina), y decimos que un tablero es **bueno** si todas sus casillas son buenas (incluso aquellas que estén vacías).

El objetivo es construir un tablero bueno, de tamaño 100×100, que maximice el número de piezas colocadas.

Entrada y salida

Nótese que no se recibe entrada.

Este problema solo tiene un caso. Debes imprimir el tablero bueno que has construido. Escribe 100 líneas cada una con 100 enteros separados por espacios. Si en tu tablero hay una pieza en la casilla (i, j), escribe un `1` en la j-ésima posición de la línea i. Si no, escribe un `0` en esa posición.

Una posible salida con un tablero bueno, si nos pidieran un tablero de dimensiones 2×2, sería:

```
1 1
1 0
```

Subtareas

Si el tablero escrito no es bueno, recibirás 0 puntos. En caso contrario, si has conseguido colocar x piezas, recibirás una puntuación acorde a la siguiente función:

$$f(x) = \begin{cases} x, & \text{si } 0 \leq x < 15 \\ 15 + (x - 15) \cdot \frac{85}{3132}, & \text{si } 15 \leq x < 3147 \\ 100, & \text{si } x \geq 3147 \end{cases}$$

Algunos puntos interesantes de la función se muestran en la siguiente tabla:

Piezas	Puntos
15	15
1215	≈ 48
2500	≈ 82
2800	≈ 90
2930	≈ 94
3147	100

Festival de música

MUSICA

En un festival de música se van a reproducir n canciones en orden aleatorio. Algunas canciones animan más al público que otras; en concreto, se sabe que el público le da una nota a_i a la i-ésima canción.

Decimos que se produce un *subidón* cuando se reproduce una canción que es mejor que la anterior según el público; es decir, si las canciones se reproducen siguiendo una permutación de índices p (donde p es una permutación de $\{1, 2, \ldots, n\}$), decimos que la posición i-ésima (para $2 \leq i \leq n$) es un subidón si $a_{p_i} > a_{p_{i-1}}$. Decimos que una reproducción de canciones tiene k subidones si existen exactamente k posiciones que son un subidón.

Dado k, cuenta cuántas permutaciones p existen tales que, si se reproducen las canciones siguiendo la permutación p (primero a_{p_1}, después a_{p_2}, etc.), hay exactamente k subidones. Como este número puede ser muy grande, da la respuesta módulo $10^9 + 7$.

Entrada y salida

La primera línea de la entrada contiene el número de casos T.

Por cada caso, habrá una línea de entrada con los enteros n y k.

La siguiente línea de cada caso contiene n enteros a_i, las notas del público para las n canciones.

Para cada caso, debes imprimir un entero: el número de permutaciones, módulo $10^9 + 7$, que producen exactamente k subidones.

Restricciones

- $1 \leq T \leq 2000$
- $1 \leq n \leq 2000$
- $0 \leq k \leq n - 1$
- La suma de n para todos los casos es como mucho 2000.
- $1 \leq a_i \leq 10^9$

Subtareas

1. (8 puntos) La suma de n para todos los casos es como mucho 9.
2. (21 puntos) $k = 0$.
3. (32 puntos) a es una permutación de $\{1, 2, \ldots, n\}$.
4. (39 puntos) Sin restricciones adicionales.

Ejemplo

Entrada	Salida
3 3 1 1 2 3 4 2 3 3 2 2 5 4 3 3 3 3 3	4 4 0

En el **primer caso**, tenemos 3 canciones con notas 1, 2 y 3 y queremos conseguir exactamente 1 subidón. Existen 6 permutaciones posibles de los índices. Si analizamos las secuencias de notas que generan:

- Si reproducimos las canciones en el orden de índices [1, 2, 3], las notas son 1, 2, 3. Hay 2 subidones (de 1 a 2, y de 2 a 3).
- Si el orden es [3, 2, 1], las notas son 3, 2, 1. Hay 0 subidones.
- Los 4 órdenes restantes generan exactamente 1 subidón. Por ejemplo, el orden de índices [1, 3, 2] genera las notas 1, 3, 2, donde solo el salto de 1 a 3 es un subidón. Las otras válidas son [2, 1, 3], [2, 3, 1] y [3, 1, 2]. Por lo tanto, la respuesta es 4.

En el **segundo caso**, tenemos 4 canciones con notas 3, 3, 2, 2 y queremos exactamente 2 subidones. Para conseguir 2 subidones con estas notas, la única estructura válida para la secuencia reproducida es alternar las notas bajas y altas: 2, 3, 2, 3. Como permutar los índices importa, las 4 permutaciones de índices válidas que generan esta secuencia de notas son: [3, 1, 4, 2], [3, 2, 4, 1], [4, 1, 3, 2] y [4, 2, 3, 1].

En el **tercer caso**, tenemos 5 canciones y todas tienen la misma nota (3). Queremos conseguir 4 subidones. Como la condición para un subidón es que la nota de la canción sea *estrictamente mayor* que la anterior ($a_{p_i} > a_{p_{i-1}}$), es imposible conseguir un subidón si todas las notas son iguales. El número de subidones siempre será 0, por lo que existen 0 permutaciones que cumplan lo que se pide.