Problemi difficili (`problemset`)

La Giornata dell'Informatica è ormai vicina e servono nuovi problemi. Francesco si mette al lavoro ma ben presto si accorge che i problemi che ha inventato sono troppo difficili e neanche lui è in grado di risolverli!

Per scongiurare l'odio dei partecipanti vuole scegliere i più facili e per farlo deve quindi ordinarli per difficoltà.

Per riuscire a stabilire il più oggettivamente possibile quest'ordine, Francesco decide di basarsi su dati reali, facendo provare la gara da N *tester*, numerati da 0 a $N-1$.

Figura 1: Gli innumerevoli *tester* di Francesco.

Ogni tester ha a disposizione 3 ore per provare a risolvere gli M problemi, numerati da 0 a $M-1$.

Al termine della prova, l'i-esimo tester ha risolto C_i problemi $S_{i,0}, S_{i,1}, ..., S_{i,C_i-1}$.

Francesco non vuole che un problema A venga prima di un problema B nell'ordinamento se almeno un tester ha risolto B ma non ha risolto A, perché questo significherebbe che per quel tester A è più difficile di B.

Aiuta Francesco a trovare un ordinamento valido dei problemi, oppure a stabilire che non esiste!

Implementazione

Dovrai sottoporre un unico file, con estensione `.cpp`, `.py`, `.cs` o `.java`.

> ⇨ Tra gli allegati di questo task troverai dei template `problemset.*` con un esempio di implementazione.

Dovrai implementare la seguente funzione:

C++	`vector<int> bilancia(int N, int M, vector<vector<int>> S);`
Python	`def bilancia(N: int, M: int, S: list[list[int]]) -> list[int]:`

Java	`public static int[] bilancia(int N, int M, int[][] S)`
C#	`public static int[] bilancia(int N, int M, int[][] S)`

- L'intero N rappresenta il numero di tester.
- L'intero M rappresenta il numero di problemi.
- L'array S, indicizzato da 0 a $N-1$, contiene i problemi risolti da ciascun tester. In particolare, per ogni $0 \leq i < N$, S_i è un array indicizzato da 0 a $C_i - 1$ contenente i problemi risolti dall'i-esimo tester.
- La funzione deve restituire un array di lunghezza M contenente un possibile ordinamento dei problemi, oppure un array vuoto se quest'ordine non esiste.

Grader di prova

Nella directory relativa a questo problema è presente una versione semplificata del grader usato durante la correzione, che puoi usare per testare le tue soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama la funzione che devi implementare e scrive su `stdout`, secondo il seguente formato.

Il file di input è composto da $N + 1$ righe, contenenti:

- Riga 1: gli interi N e M.
- Riga $2 + i$ ($0 \leq i < N$): L'intero C_i seguito da C_i interi $S_{i,j}$.

Il file di output è composto da un'unica riga, contenente la stringa «IMPOSSIBLE» se `bilancia` restituisce un array vuoto, i valori restituiti altrimenti.

Assunzioni

- $1 \leq N \leq 100\,000$.
- $1 \leq M \leq 100\,000$.
- $0 \leq C_i < M$ per ogni $0 \leq i < N$.
- $C_0 + C_1 + ... + C_{N-1} \leq 1\,000\,000$.
- $0 \leq S_{i,j} < M$ per ogni $0 \leq i < N$, $0 \leq j < C_i$.
- Nessun tester risolve più volte lo stesso problema.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test che lo compongono.

- **Subtask 1 [0 punti]**: Casi d'esempio.
- **Subtask 2 [16 punti]**: $M \leq 2$.
- **Subtask 3 [12 punti]**: $N \leq 2$.
- **Subtask 4 [21 punti]**: $N \leq 10$, $M \leq 10$.
- **Subtask 5 [28 punti]**: $N \leq 1000$, $M \leq 1000$. Inoltre $C_0 + C_1 + ... + C_{N-1} \leq 1000$.
- **Subtask 6 [23 punti]**: No additional constraint.

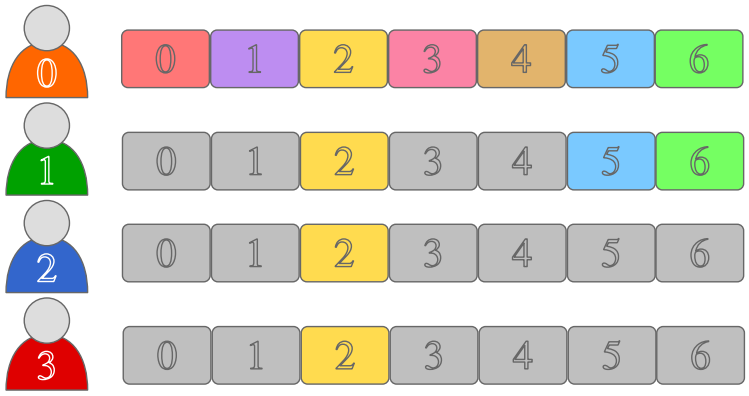
Esempi di input/output

stdin	stdout
4 7 7 4 3 0 5 1 6 2 3 2 6 5 1 2 1 2	2 5 6 3 1 4 0

stdin	stdout
4 6 6 1 3 4 5 0 2 2 1 2 0 4 1 5 0 4	IMPOSSIBLE

Spiegazione

Il **primo caso d'esempio** può essere schematizzato nel seguente modo:



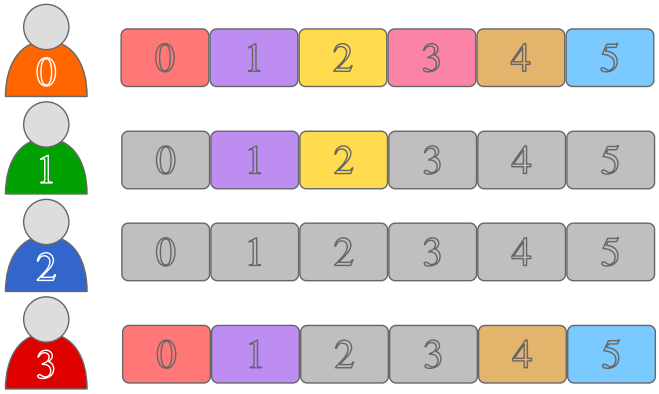
Una possibile soluzione è $[2, 5, 6, 3, 1, 4, 0]$:

Solo il problema 2 è stato risolto da tutti, dunque è sicuramente il più facile. I problemi 5 e 6 sono stati risolti dai *tester* 0 e 1 quindi in saranno, in qualche ordine, il secondo e il terzo più facili. Gli altri problemi sono stati risolti solo dal *tester* 0, di conseguenza sono gli ultimi nella lista, in qualche ordine.

Possiamo vedere che per ogni coppia di problemi A e B con A più difficile di B nel nostro ordinamento, se un tester ha risolto A, allora ha risolto anche B. La soluzione è quindi valida!

Un altra soluzione valida è $[2, 6, 5, 0, 4, 3, 1]$ per il motivo sopra descritto.

Nel **secondo caso di esempio** non è possibile trovare una soluzione valida.



Supponiamo che il problema 2 sia più facile del problema 5, allora il *tester* 3, che ha risolto il problema 5, avrebbe dovuto risolvere anche il problema 2.

Vice versa, se il problema 2 fosse più difficile del problema 5, il *tester* 1, che ha risolto il problema 2, avrebbe dovuto risolvere anche il problema 5.

Nessuna delle due opzioni è valida, quindi non esiste un ordinamento che rispetti le condizioni richieste.