Biglietti di Carnevale (tickets)

Ringo ha dei biglietti premio del Carnevale di Singapore che vuole riscuotere. Ogni biglietto è di uno tra n possibili colori (dove n è **pari**) ed ha riportato un intero non-negativo (interi su biglietti diversi potrebbero coincidere). Ringo ha m biglietti di ogni colore, per un totale di $n \cdot m$ biglietti. Il biglietto j di colore i ha riportato l'intero $x[i][j]$ ($0 \le i \le n-1, 0 \le j \le m-1$).

Il procedimento di riscossione dei premi è diviso in k round, numerati da 0 a $k-1$, in ciascuno dei quali:

- Ringo sceglie un biglietto per ogni colore, sui quali sono riportati gli interi $a[0], a[1] \ \ldots \ a[n-1]$ (in un qualunque ordine), e li dà al conduttore della riscossione;
- Il conduttore estrae un intero b da un'urna speciale, e quindi calcola le differenze assolute tra $a[i]$ e b per ogni $i = 0 \ldots n-1$;
- Per questo round, il conduttore dà a Ringo un premio di valore pari alla somma S di queste differenze assolute, per poi eliminare per sempre i biglietti usati.

Eventuali biglietti rimasti dopo k round non possono più essere utilizzati.

Controllando bene, Ringo si è accorto che il gioco è truccato! L'urna speciale in realtà contiene una stampante, che produce ogni volta l'intero b che minimizza il valore del premio per quel round. Sapendolo, aiuta Ringo a scegliere i biglietti da consegnare nei k round, di modo da massimizzare il valore totale di tutti i premi ottenuti.

Note di implementazione

Devi implementare la seguente funzione:

```
int64 find_maximum(int k, int[][] x)
```

- k: il numero di round.
- x: un array $n \times m$ contenente gli interi riportati sui biglietti. Per ogni colore, gli interi sono ordinati in modo non-decrescente.
- Questa funzione è chiamata esattamente una volta.
- Questa funzione deve fare esattamente una chiamata ad `allocate_tickets` (vedi sotto), descrivendo le scelte di n biglietti per ognuno dei k round che massimizzano il valore totale dei premi.
- Questa funzione deve restituire il massimo valore totale dei premi.

La funzione `allocate_tickets` è definita come segue:

```
void allocate_tickets(int[][] s)
```

- s: un array $n \times m$, dove $s[i][j] = r$ se il biglietto j di colore i è usato nel round r, oppure -1 se non è usato affatto.
- Per ogni $0 \leq i \leq n-1$, i valori $0, 1, 2, \ldots, k-1$ devono comparire esattamente una volta in $s[i][0], s[i][1], \ldots, s[i][m-1]$, e tutti gli altri valori devono essere -1.
- Se ci sono più modi di scegliere i biglietti che risultano nello stesso massimo valore totale, è ammesso riportare una qualunque di esse.

Esempi

Esempio 1

Considera la seguente chiamata:

```
find_maximum(2, [[0, 2, 5],[1, 1, 3]])
```

Questo significa che:

- ci sono $k = 2$ round;
- sui biglietti di colore 0 sono riportati gli interi 0, 2 e 5;
- sui biglietti di colore 1 sono riportati gli interi 1, 1 e 3.

Una possibile selezione che ottiene il massimo valore totale è:

- Nel round 0, scegliere il biglietto 0 di colore 0 (con intero 0) e il biglietto 2 di colore 1 (con intero 3). Un valore b che minimizza il premio in questo caso è $b = 1$, per un premio di $|1-0| + |1-3| = 1 + 2 = 3$.
- Nel round 1, scegliere il biglietto 2 di colore 0 (con intero 5) e il biglietto 1 di colore 1 (con intero 1). Un valore b che minimizza il premio in questo round è $b = 3$, per un premio di $|3-1| + |3-5| = 2 + 2 = 4$.
- Quindi, il valore totale ottenuto è $3 + 4 = 7$.

Per riportare questa soluzione, la funzione `find_maximum` deve fare la seguente chiamata:

- `allocate_tickets([[0, -1, 1], [-1, 1, 0]])`

e poi restituire il valore 7.

Esempio 2

Considera la seguente chiamata:

```
find_maximum(1, [[5, 9], [1, 4], [3, 6], [2, 7]])
```

Questo significa che:

- c'è un solo round,
- sui biglietti di colore 0 sono riportati gli interi 5 e 9;
- sui biglietti di colore 1 sono riportati gli interi 1 e 4;
- sui biglietti di colore 2 sono riportati gli interi 3 e 6;
- sui biglietti di colore 3 sono riportati gli interi 2 e 7.

Una possibile selezione che ottiene il massimo valore totale è:

- Nel round 0, scegliere il biglietto 1 di colore 0 (con intero 9), il biglietto 0 di colore 1 (con intero 1), il biglietto 0 di colore 2 (con intero 3), e il biglietto 1 di colore 3 (con intero 7). Un valore b che minimizza il premio in questo caso è $b = 3$, per un premio di $|3-9| + |3-1| + |3-3| + |3-7| = 6+2+0+4 = 12$.

Per riportare questa soluzione, la funzione `find_maximum` deve fare la seguente chiamata:

- `allocate_tickets([[-1, 0], [0, -1], [0, -1], [-1, 0]])`

e poi restituire il valore 12.

Assunzioni

- $2 \leq n \leq 1500$, ed n è pari.
- $1 \leq k \leq m \leq 1500$
- $0 \leq x[i][j] \leq 10^9$ (per ogni $0 \leq i \leq n-1$, $0 \leq j \leq m-1$)
- $x[i][j-1] \leq x[i][j]$ (per ogni $0 \leq i \leq n-1$, $1 \leq j \leq m-1$)

Subtask

1. (11 punti) $m = 1$
2. (16 punti) $k = 1$
3. (14 punti) $0 \leq x[i][j] \leq 1$ (per ogni $0 \leq i \leq n-1$, $0 \leq j \leq m-1$)
4. (14 punti) $k = m$
5. (12 punti) $n, m \leq 80$
6. (23 punti) $n, m \leq 300$
7. (10 punti) Nessuna limitazione aggiuntiva.

Grader di esempio

Il grader di esempio legge l'input nel seguente formato:

- riga 1: n m k
- righe $2 + i$ ($0 \le i \le n - 1$): $x[i][0]$ $x[i][1]$ $\ldots$ $x[i][m-1]$

Il grader di esempio stampa l'output nel seguente formato:

- riga 1: il valore restituito da `find_maximum`
- righe $2 + i$ ($0 \le i \le n - 1$): $s[i][0]$ $s[i][1]$ $\ldots$ $s[i][m-1]$