



Carnival Tickets (tickets)

Ringo is at a carnival in Singapore. He has some prize tickets in his bag, which he would like to use at the prize game stall. Each ticket comes in one of n colours and has a non-negative integer printed on it. The integers printed on different tickets might be the same. Due to a quirk in the carnival rules, n is guaranteed to be **even**.

Ringo has m tickets of each colour in his bag, that is a total of $n \cdot m$ tickets. The ticket j of the colour i has the integer $x[i][j]$ printed on it ($0 \leq i \leq n - 1$ and $0 \leq j \leq m - 1$).

The prize game is played in k rounds, numbered from 0 to $k - 1$. Each round is played in the following order:

- From his bag, Ringo selects a **set** of n tickets, one ticket from each colour. He then gives the set to the game master.
- The game master notes down the integers $a[0], a[1] \dots a[n - 1]$ printed on the tickets of the set. The order of these n integers is not important.
- The game master pulls out a special card from a lucky draw box and notes down the integer b printed on that card.
- The game master calculates the absolute differences between $a[i]$ and b for each i from 0 to $n - 1$. Let S be the sum of these absolute differences.
- For this round, the game master gives Ringo a prize with a value equal to S .
- The tickets in the set are discarded and cannot be used in future rounds.

The remaining tickets in Ringo's bag after k rounds of the game are discarded.

By watching closely, Ringo realized that the prize game is rigged! There is actually a printer inside the lucky draw box. In each round, the game master finds an integer b that minimizes the value of the prize of that round. The value chosen by the game master is printed on the special card for that round.

Having all this information, Ringo would like to allocate tickets to the rounds of the game. That is, he wants to select the ticket set to use in each round in order to maximize the total value of the prizes.

Implementation details

You should implement the following procedure:

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

- k : the number of rounds.

- x : an $n \times m$ array describing the integers on each ticket. Tickets of each colour are sorted in non-decreasing order of their integers.
- This procedure is called exactly once.
- This procedure should make exactly one call to `allocate_tickets` (see below), describing k ticket sets, one for each round. The allocation should maximize the total value of the prizes.
- This procedure should return the maximum total value of the prizes.

The procedure `allocate_tickets` is defined as follows:

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

- s : an $n \times m$ array. The value of $s[i][j]$ should be r if the ticket j of the colour i is used in the set of round r of the game, or -1 if it is not used at all.
- For each $0 \leq i \leq n - 1$, among $s[i][0], s[i][1], \dots, s[i][m - 1]$ each value $0, 1, 2, \dots, k - 1$ must occur exactly once, and all other entries must be -1 .
- If there are multiple allocations resulting in the maximum total prize value, it is allowed to report any of them.

Examples

Example 1

Consider the following call:

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

This means that:

- there are $k = 2$ rounds;
- the integers printed on the tickets of colour 0 are 0, 2 and 5, respectively;
- the integers printed on the tickets of colour 1 are 1, 1 and 3, respectively.

A possible allocation that gives the maximum total prize value is:

- In round 0, Ringo picks ticket 0 of colour 0 (with the integer 0) and ticket 2 of colour 1 (with the integer 3). The lowest possible value of the prize in this round is 3. E.g., the game master may choose $b = 1$: $|1 - 0| + |1 - 3| = 1 + 2 = 3$.
- In round 1, Ringo picks ticket 2 of colour 0 (with the integer 5) and ticket 1 of colour 1 (with the integer 1). The lowest possible value of the prize in this round is 4. E.g., the game master may choose $b = 3$: $|3 - 1| + |3 - 5| = 2 + 2 = 4$.
- Therefore, the total value of the prizes would be $3 + 4 = 7$.

To report this allocation, the procedure `find_maximum` should make the following call to `allocate_tickets`:

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

Finally, the procedure `find_maximum` should return 7.

Example 2

Consider the following call:

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

This means that:

- there is only one round,
- the integers printed on the tickets of colour 0 are 5 and 9, respectively;
- the integers printed on the tickets of colour 1 are 1 and 4, respectively;
- the integers printed on the tickets of colour 2 are 3 and 6, respectively;
- the integers printed on the tickets of colour 3 are 2 and 7, respectively.

A possible allocation that gives the maximum total prize value is:

- In round 0, Ringo picks ticket 1 of colour 0 (with the integer 9), ticket 0 of colour 1 (with the integer 1), ticket 0 of colour 2 (with the integer 3), and ticket 1 of colour 3 (with the integer 7). The lowest possible value of the prize in this round is 12, when the game master chooses $b = 3$: $|3 - 9| + |3 - 1| + |3 - 3| + |3 - 7| = 6 + 2 + 0 + 4 = 12$.

To report this solution, the procedure `find_maximum` should make the following call to `allocate_tickets`:

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

Finally, the procedure `find_maximum` should return 12.

Constraints

- $2 \leq n \leq 1500$ and n is even.
- $1 \leq k \leq m \leq 1500$
- $0 \leq x[i][j] \leq 10^9$ (for all $0 \leq i \leq n - 1$ and $0 \leq j \leq m - 1$)
- $x[i][j - 1] \leq x[i][j]$ (for all $0 \leq i \leq n - 1$ and $1 \leq j \leq m - 1$)

Subtasks

1. (11 points) $m = 1$
2. (16 points) $k = 1$
3. (14 points) $0 \leq x[i][j] \leq 1$ (for all $0 \leq i \leq n - 1$ and $0 \leq j \leq m - 1$)
4. (14 points) $k = m$
5. (12 points) $n, m \leq 80$

6. (23 points) $n, m \leq 300$
7. (10 points) No additional constraints.

Sample grader

The sample grader reads the input in the following format:

- line 1: $n \ m \ k$
- line $2 + i$ ($0 \leq i \leq n - 1$): $x[i][0] \ x[i][1] \ \dots \ x[i][m - 1]$

The sample grader prints your answer in the following format:

- line 1: the return value of `find_maximum`
- line $2 + i$ ($0 \leq i \leq n - 1$): $s[i][0] \ s[i][1] \ \dots \ s[i][m - 1]$