Aliens

Our satellite has just discovered an alien civilization on a remote planet. We have already obtained a low-resolution photo of a square area of the planet. The photo shows many signs of intelligent life. Our experts have identified n points of interest in the photo. We now want to take high-resolution photos that contain all of those n points.

Internally, the satellite has divided the area of the low-resolution photo into an m by m grid of unit square cells. Both rows and columns of the grid are consecutively numbered from 0 to $m-1$ (from the top and left, respectively). We use (i, j) to denote the cell in row i and column j. Each point of interest is contained inside one cell. Each cell may contain an arbitrary number of these points.

Our satellite is on a stable orbit that passes directly over the *main* diagonal of the grid. The main diagonal is the diagonal which passes through cells (i, i), for all $0 \leq i \leq m-1$. The satellite can take a high-resolution photo of any area that satisfies the following constraints:

- the shape of the area is a square,
- one diagonal of the square is fully contained in the main diagonal of the grid,
- each cell of the grid is either completely inside or completely outside the photographed area.
  The satellite is able to take at most k high-resolution photos.

Once the satellite is done taking photos, it will transmit the high-resolution photo of each photographed cell to our home base (regardless of whether that cell contains some points of interest). Each photographed cell will be transmitted only *once*, even if it was photographed several times.

Thus, we have to choose at most k square areas that will be photographed, assuring that:

- each cell containing at least one point of interest is photographed at least once, and
- the number of cells that are photographed at least once is minimized.
  Your task is to find this number.

Implementation details

You should implement the following function (method):

- `int64 take_photos(int n, int m, int k, int[] r, int[] c)`
  - `n`: the number of points of interest,
  - `m`: the number of rows (and also columns) in the grid,
  - `k`: the maximum number of photos the satellite can take,

- `r` and `c`: two arrays of length n describing the coordinates of the grid cells that contain points of interest. For $0 \leq i \leq n-1$, the i-th point of interest is located in the cell (`r[i]`, `c[i]`),
- the function should return the smallest possible total number of cells that are photographed at least once (given that the photo must cover all points of interest).

Please use the provided template files for details of implementation in your programming language.

Examples

Example 1

```
take_photos(5, 7, 2, [0, 4, 4, 4, 4], [3, 4, 6, 5, 6])
```

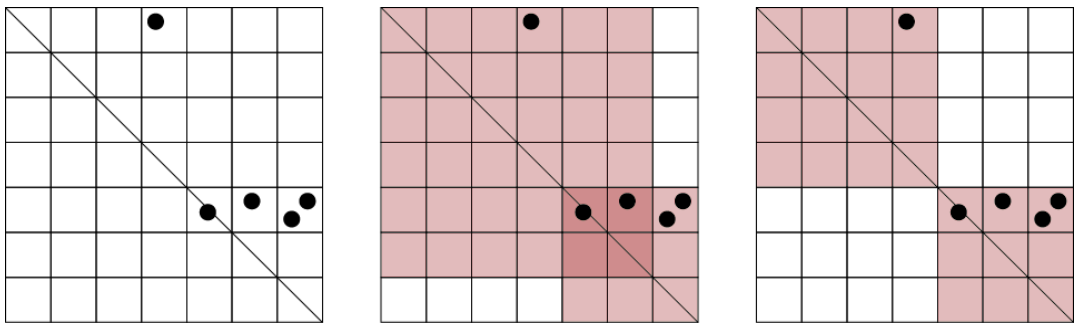
In this example we have a 7×7 grid with 5 points of interest. The points of interest are located in four different cells: $(0, 3)$, $(4, 4)$, $(4, 5)$ and $(4, 6)$. You may take at most 2 high-resolution photos.

One way to capture all five points of interest is to make two photos: one with the cells $(0, 0)$ and $(5, 5)$ in the opposite corners, and the other with cells $(4, 4)$ and $(6, 6)$ in the opposite corners. If we take these two photos, the satellite will transmit the photos of 41 cells. This amount is not optimal.

The optimal solution uses one photo to capture the 4×4 square containing cells $(0, 0)$ and $(3, 3)$ and another photo to capture the 3×3 square containing cells $(4, 4)$ and $(6, 6)$. This results in only 25 photographed cells, which is optimal, so `take_photos` should return 25.

Note that it is sufficient to photograph the cell $(4, 6)$ once, even though it contains two points of interest.

Both ways of taking the photos are depicted below. The figure on the right shows the optimal solution.

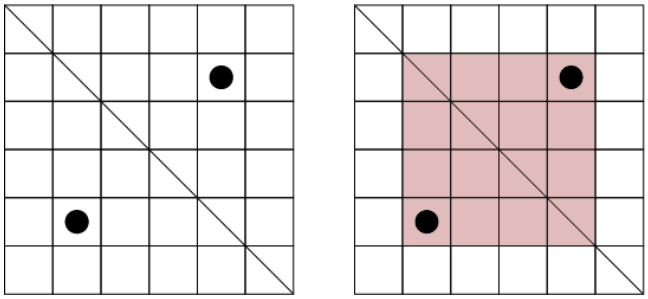


Example 2

```
take_photos(2, 6, 2, [1, 4], [4, 1])
```

Here we have 2 points of interest located symmetrically: in the cells $(1, 4)$ and

$(4, 1)$. Any valid photo that contains one of them contains the other one as well. Therefore, it is sufficient to use a single photo. The optimal solution (depicted below) captures the photo of 16 cells.



Subtasks

For all subtasks, $1 \le k \le n$.

1. (4 points) $1 \le n \le 50$, $1 \le m \le 100$, $k = n$.
2. (12 points) $1 \le n \le 500$, $1 \le m \le 1000$, for all i such that $0 \le i \le n - 1$, $r_i = c_i$.
3. (9 points) $1 \le n \le 500$, $1 \le m \le 1000$,
4. (16 points) $1 \le n \le 2000$, $1 \le m \le 1\,000\,000$,
5. (12 points) $1 \le n \le 20\,000$, $1 \le k \le 100$, $1 \le m \le 1\,000\,000$,
6. (7 points) $1 \le n \le 20\,000$, $1 \le k \le 1000$, $1 \le m \le 1\,000\,000$,
7. (40 points) $1 \le n \le 200\,000$, $1 \le m \le 1\,000\,000$.

Sample grader

The sample grader reads the input in the following format:

- line 1: integers n, m and k,
- line 2 + i ($0 \le i \le n - 1$): integers r_i and c_i.