Broken Line

Azerbaijan is famous for its carpets. As a master carpet designer you want to make a new design by drawing a **broken line**. A broken line is a sequence of t line segments in a two-dimensional plane, which is defined by a sequence of $t+1$ points $p_0, \ldots, p_t$ as follows. For each $0 \leq j \leq t-1$ there is a segment connecting points p_j and p_{j+1}.

In order to make the new design, you have already marked n **dots** in a two-dimensional plane. The coordinates of dot i ($1 \leq i \leq n$) are $(x[i], y[i])$. **No two dots have the same x or the same y coordinate.**

You now want to find a sequence of points $(sx[0], sy[0]), (sx[1], sy[1]), \ldots, (sx[k], sy[k])$, which defines a broken line that

- starts at $(0, 0)$ (that is, $sx[0] = 0$ and $sy[0] = 0$),
- contains all of the dots (not necessarily as the endpoints of the segments), and
- consists solely of horizontal or vertical segments (two consecutive points defining the broken line have an equal x or y coordinate).

The broken line is allowed to intersect or overlap itself in any way. Formally, each point of the plane may belong to any number of segments of the broken line.

This is an output-only task with partial scoring. You are given 10 input files specifying the locations of dots. For each input file, you should submit an output file describing a broken line with the required properties. For each output file that describes a valid broken line your score depends on the **number of segments** in the broken line (see Scoring below).

You are not supposed to submit any source code for this task.

Input format

Each input file is in the following format:

- line 1: n
- line $1+i$ (for $1 \leq i \leq n$): $x[i]$ $y[i]$

Output format

Each output file must be in the following format:

- line 1: k
- line $1+j$ (for $1 \le j \le k$): $sx[j]$ $sy[j]$

Note that the second line should contain $sx[1]$ and $sy[1]$ (i.e., the output **should not** contain $sx[0]$ and $sy[0]$). Each $sx[j]$ and $sy[j]$ should be an integer.

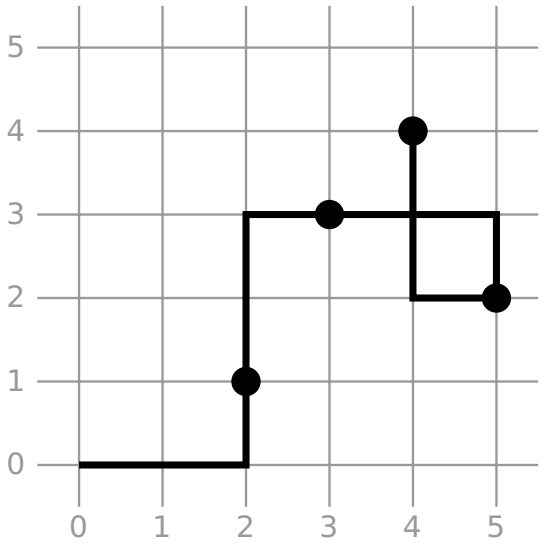
Example

For the sample input:

```
4
2 1
3 3
4 4
5 2
```

a possible valid output is:

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



Please note this example is not among the actual inputs of this task.

Constraints

- $1 \le n \le 100\,000$
- $1 \le x[i], y[i] \le 10^9$
- All values of $x[i]$ and $y[i]$ are integers.
- No two dots have the same x or the same y coordinates, i.e. $x[i_1] \neq x[i_2]$ **and** $y[i_1] \neq y[i_2]$ for $i_1 \neq i_2$.
- $-2 \cdot 10^9 \le sx[j], sy[j] \le 2 \cdot 10^9$
- The size of each submitted file (either an output or a zipped file) cannot exceed 15MB.

Scoring

For each test case, you can get up to 10 points. Your output for a test case will get 0 points if it does not specify a broken line with the required properties. Otherwise, the score will be determined using a decreasing sequence $c_1, \ldots, c_{10}$, which varies by testcase.

Assume that your solution is a valid broken line consisting of k segments. Then, you will get

- i points, if $k = c_i$ (for $1 \le i \le 10$),
- $i + \frac{c_i - k}{c_i - c_{i+1}}$ points, if $c_{i+1} < k < c_i$ (for $1 \le i \le 9$),
- 0 points, if $k > c_1$,
- 10 points, if $k < c_{10}$.

The sequence $c_1, \ldots, c_{10}$ for each testcase is given below.

Testcases	01	02	03	04	05	06	07-10
n	20	600	5 000	50 000	72 018	91 891	100 000
c_1	50	1 200	10 000	100 000	144 036	183 782	200 000
c_2	45	937	7 607	75 336	108 430	138 292	150 475
c_3	40	674	5 213	50 671	72 824	92 801	100 949
c_4	37	651	5 125	50 359	72 446	92 371	100 500
c_5	35	640	5 081	50 203	72 257	92 156	100 275
c_6	33	628	5 037	50 047	72 067	91 941	100 050
c_7	28	616	5 020	50 025	72 044	91 918	100 027
c_8	26	610	5 012	50 014	72 033	91 906	100 015
c_9	25	607	5 008	50 009	72 027	91 900	100 009
c_{10}	23	603	5 003	50 003	72 021	91 894	100 003

Visualizer

In the attachments of this task, there is a script that allows you to visualize input and output files.

To visualize an input file, use the following command:

```
python vis.py [input file]
```

You can also visualize your solution for some input, using the following command. Due to technical limitations, the provided visualizer shows only **the first** 1000 **segments** of the output file.

```
python vis.py [input file] --solution [output file]
```

Example:

```
python vis.py examples/00.in --solution examples/00.out
```

Vision Program

You are implementing a vision program for a robot. Each time the robot camera takes a picture, it is stored as a black and white image in the robot's memory. Each image is an $H \times W$ grid of pixels, with rows numbered 0 through $H-1$ and columns numbered 0 through $W-1$. There are **exactly two** black pixels in each image, and all other pixels are white.

The robot can process each image with a program consisting of simple instructions. You are given the values of H, W, and a positive integer K. Your goal is to write a procedure to produce a program for the robot that, for any image, determines whether the **distance** between the two black pixels is exactly K. Here, the distance between a pixel in row r_1 and column c_1 and a pixel in row r_2 and column c_2 is $|r_1 - r_2| + |c_1 - c_2|$. In this formula $|x|$ denotes the absolute value of x, which equals x if $x \geq 0$ and equals $-x$ if $x < 0$.

We now describe how the robot works.

The robot's memory is a sufficiently large array of cells, indexed from 0. Each cell can store either 0 or 1 and its value, once set, will not be changed. The image is stored row by row in cells indexed 0 through $H \cdot W - 1$. The first row is stored in cells 0 through $W-1$, and the last row is stored in cells $(H-1) \cdot W$ through $H \cdot W - 1$. In particular, if the pixel in row i and column j is black, the value of cell $i \cdot W + j$ is 1, otherwise it is 0.

A robot's program is a sequence of **instructions**, which are numbered with consecutive integers starting from 0. When the program is run, the instructions are executed one by one. Each instruction reads the values of one or more cells (we call these values the instruction's **inputs**) and produces a single value equal to 0 or 1 (we call this value the instruction's **output**). The output of instruction i is stored in cell $H \cdot W + i$. The inputs of instruction i can only be cells that store either pixels or outputs of previous instructions, i.e. cells 0 to $H \cdot W + i - 1$.

There are four types of instructions:

- `NOT`: has exactly one input. Its output is 1 if the input is 0, otherwise its output is 0.
- `AND`: has one or more inputs. Its output is 1 if and only if **all** of the inputs are 1.
- `OR`: has one or more inputs. Its output is 1 if and only if **at least one** of the inputs is 1.
- `XOR`: has one or more inputs. Its output is 1 if and only if an **odd number** of the

inputs are 1.

The output of the last instruction of the program should be 1 if the distance between the two black pixels is exactly K, and 0 otherwise.

Implementation details

You should implement the following procedure:

```
void construct_network(int H, int W, int K)
```

- H, W: dimensions of each image taken by the robot's camera
- K: a positive integer
- This procedure should produce a robot's program. For any image taken by the robot's camera, this program should determine whether the distance between the two black pixels in the image is exactly K.

This procedure should call one or more of the following procedures to append instructions to the robot's program (which is initially empty):

```
int add_not(int N)
int add_and(int[] Ns)
int add_or(int[] Ns)
int add_xor(int[] Ns)
```

- Append a `NOT`, `AND`, `OR`, or `XOR` instruction, respectively.
- N (for `add_not`): the index of the cell from which the appended `NOT` instruction reads its input
- Ns (for `add_and`, `add_or`, `add_xor`): array containing the indices of the cells from which the appended `AND`, `OR`, or `XOR` instruction reads its inputs
- Each procedure returns the index of the cell that stores the output of the instruction. The consecutive calls to these procedures return consecutive integers starting from $H \cdot W$.

The robot's program can consist of at most $10\,000$ instructions. The instructions can read at most $1\,000\,000$ values in total. In other words, the total length of Ns arrays in all calls to `add_and`, `add_or` and `add_xor` plus the number of calls to `add_not` cannot exceed $1\,000\,000$.

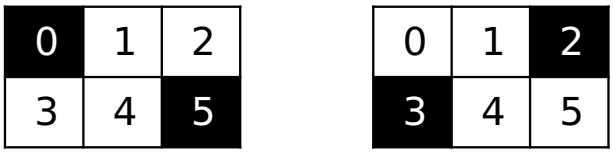
After appending the last instruction, procedure `construct_network` should return. The robot's program will then be evaluated on some number of images. Your solution passes a given test case if for each of these images, the output of the last instruction is 1 if and only if the distance between the two black pixels in the image is equal to K.

The grading of your solution may result in one of the following error messages:

- `Instruction with no inputs`: an empty array was given as the input to `add_and`, `add_or`, or `add_xor`.
- `Invalid index`: an incorrect (possibly negative) cell index was provided as the input to `add_and`, `add_or`, `add_xor`, or `add_not`.
- `Too many instructions`: your procedure attempted to add more than $10\,000$ instructions.
- `Too many inputs`: the instructions read more than $1\,000\,000$ values in total.

Example

Assume $H = 2$, $W = 3$, $K = 3$. There are only two possible images where the distance between the black pixels is 3.



- Case 1: black pixels are 0 and 5
- Case 2: black pixels are 2 and 3

A possible solution is to build a robot's program by making the following calls:

1. `add_and([0, 5])`, which adds an instruction that outputs 1 if and only if the first case holds. The output is stored in cell 6.
2. `add_and([2, 3])`, which adds an instruction that outputs 1 if and only if the second case holds. The output is stored in cell 7.
3. `add_or([6, 7])`, which adds an instruction that outputs 1 if and only if one of the cases above holds.

Constraints

- $1 \leq H \leq 200$
- $1 \leq W \leq 200$
- $2 \leq H \cdot W$
- $1 \leq K \leq H + W - 2$

Subtasks

1. (10 points) $\max(H, W) \leq 3$
2. (11 points) $\max(H, W) \leq 10$
3. (11 points) $\max(H, W) \leq 30$
4. (15 points) $\max(H, W) \leq 100$
5. (12 points) $\min(H, W) = 1$

6. (8 points) Pixel in row 0 and column 0 is black in each image.
7. (14 points) $K = 1$
8. (19 points) No additional constraints.

Sample grader

The sample grader reads the input in the following format:

- line 1: H W K
- line $2 + i$ ($i \geq 0$): $r_1[i]$ $c_1[i]$ $r_2[i]$ $c_2[i]$
- last line: -1

Each line excepting the first and the last line represents an image with two black pixels. We denote the image described in line $2 + i$ by image i. One black pixel is in row $r_1[i]$ and column $c_1[i]$ and the other one in row $r_2[i]$ and column $c_2[i]$.

The sample grader first calls `construct_network(H, W, K)`. If `construct_network` violates some constraint described in the problem statement, the sample grader prints one of the error messages listed at the end of Implementation details section and exits.

Otherwise, the sample grader produces two outputs.

First, the sample grader prints the output of the robot's program in the following format:

- line $1 + i$ ($0 \leq i$): output of the last instruction in the robot's program for image i (1 or 0).

Second, the sample grader writes a file `log.txt` in the current directory in the following format:

- line $1 + i$ ($0 \leq i$): $m[i][0]$ $m[i][1]$ $\ldots$ $m[i][c-1]$

The sequence on line $1 + i$ describes the values stored in the robot's memory cells after the robot's program is run, given image i as the input. Specifically, $m[i][j]$ gives the value of cell j. Note that the value of c (the length of the sequence) is equal to $H \cdot W$ plus the number of instructions in the robot's program.

Sky Walking

Kenan drew a plan of the buildings and skywalks along one side of the main avenue of Baku. There are n buildings numbered from 0 to $n-1$ and m skywalks numbered from 0 to $m-1$. The plan is drawn on a two-dimensional plane, where the buildings and skywalks are vertical and horizontal segments respectively.

The bottom of building i $(0 \leq i \leq n-1)$ is located at point $(x[i], 0)$ and the building has height $h[i]$. Hence, it is a segment connecting the points $(x[i], 0)$ and $(x[i], h[i])$.

Skywalk j $(0 \leq j \leq m-1)$ has endpoints at buildings numbered $l[j]$ and $r[j]$ and has a positive y-coordinate $y[j]$. Hence, it is a segment connecting the points $(x[l[j]], y[j])$ and $(x[r[j]], y[j])$.

A skywalk and a building **intersect** if they share a common point. Hence, a skywalk intersects two buildings at its two endpoints, and may also intersect other buildings in between.

Kenan would like to find the length of the shortest path from the bottom of building s to the bottom of building g, assuming that one can only walk along the buildings and skywalks, or determine that no such path exists. Note that it is not allowed to walk on the ground, i.e. along the horizontal line with y-coordinate 0.

One can walk from a skywalk into a building or vice versa at any intersection. If the endpoints of two skywalks are at the same point, one can walk from one skywalk to the other.

Your task is to help Kenan answer his question.

Implementation details

You should implement the following procedure. It will be called by the grader once for each test case.

```
int64 min_distance(int[] x, int[] h, int[] l, int[] r, int[] y,
                   int s, int g)
```

- x and h: integer arrays of length n
- l, r, and y: integer arrays of length m
- s and g: two integers

- This procedure should return the length of the shortest path between the bottom of building s and the bottom of building g, if such path exists. Otherwise, it should return -1.

Examples

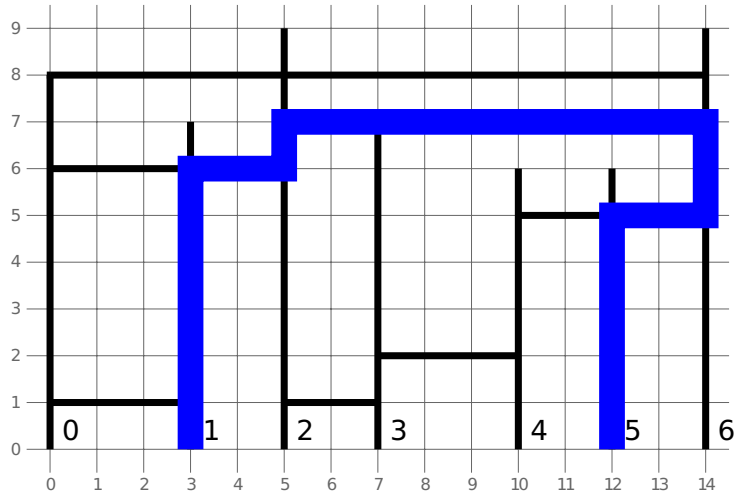
Example 1

Consider the following call:

```
min_distance([0, 3, 5, 7, 10, 12, 14],
             [8, 7, 9, 7, 6, 6, 9],
             [0, 0, 0, 2, 2, 3, 4],
             [1, 2, 6, 3, 6, 4, 6],
             [1, 6, 8, 1, 7, 2, 5],
             1, 5)
```

The correct answer is 27.

The figure below corresponds to *Example 1*:



Example 2

```
min_distance([0, 4, 5, 6, 9],
             [6, 6, 6, 6, 6],
             [3, 1, 0],
             [4, 3, 2],
             [1, 3, 6],
             0, 4)
```

The correct answer is 21.

Constraints

- $1 \leq n, m \leq 100\,000$
- $0 \leq x[0] < x[1] < \ldots < x[n-1] \leq 10^9$
- $1 \leq h[i] \leq 10^9$ (for all $0 \leq i \leq n-1$)
- $0 \leq l[j] < r[j] \leq n-1$ (for all $0 \leq j \leq m-1$)
- $1 \leq y[j] \leq \min(h[l[j]], h[r[j]])$ (for all $0 \leq j \leq m-1$)
- $0 \leq s, g \leq n-1$
- $s \neq g$
- No two skywalks have a common point, except maybe on their endpoints.

Subtasks

1. (10 points) $n, m \leq 50$
2. (14 points) Each skywalk intersects at most 10 buildings.
3. (15 points) $s = 0$, $g = n-1$, and all buildings have the same height.
4. (18 points) $s = 0$, $g = n-1$
5. (43 points) No additional constraints.

Sample grader

The sample grader reads the input in the following format:

- line 1: n m
- line $2+i$ ($0 \leq i \leq n-1$): $x[i]$ $h[i]$
- line $n+2+j$ ($0 \leq j \leq m-1$): $l[j]$ $r[j]$ $y[j]$
- line $n+m+2$: s g

The sample grader prints a single line containing the return value of `min_distance`.