icpc International Collegiate Programming Contest

The 2025 ICPC
Asia Seoul Regional Contest

Problem Set

Problem Set

Please check that you have 13 problems that span across 30 pages in total (including two cover pages).

Problem A
Adventurer Dabi
Time Limit: 3 Seconds

Dabi, the galaxy-famous adventurer, has entered a long-forgotten underground city said to hide an ancient treasure. Within its pitch-black tunnels, she must find a key and reach the treasure chest before the city collapses.

The entire city can be represented as an $h \times w$ grid ($3 \leq h, w \leq 16$). Each cell of this grid is one of the following types:

- **Empty cell**, which is a traversable passage.
- **Wall cell**, which is a solid block that cannot be entered.
- **Teleport cell**, which is an ancient device that instantly connects two distant places.

The following facts are known about this underground city:

- Every cell on the grid's boundary (i.e., topmost/bottommost row, or leftmost/rightmost column) is a wall.
- All wall cells are connected through four directions: north (up), south (down), east (right), and west (left)—that is, they form a single 4-directional connected component.
- All empty cells are connected, forming a single 4-directional connected component surrounded by walls.
- For every teleport cell, all **eight** neighboring cells are empty cells.
- Every teleport cell belongs to exactly one teleport pair. If Dabi steps into a teleport cell from any direction, she is immediately transported to its paired cell and then moves **one additional step** in the same direction she entered the teleport cell. This process never triggers another teleportation.
- Each teleport pair is labeled with one of the uppercase letters `A`, `B`, `C`, `D`, `E`, and `F`. Therefore, there are at most $6 \times 2 = 12$ teleport cells in total.
- There is exactly one **key** and one **treasure chest**, each placed on distinct empty cells, but neither is at Dabi's initial position.

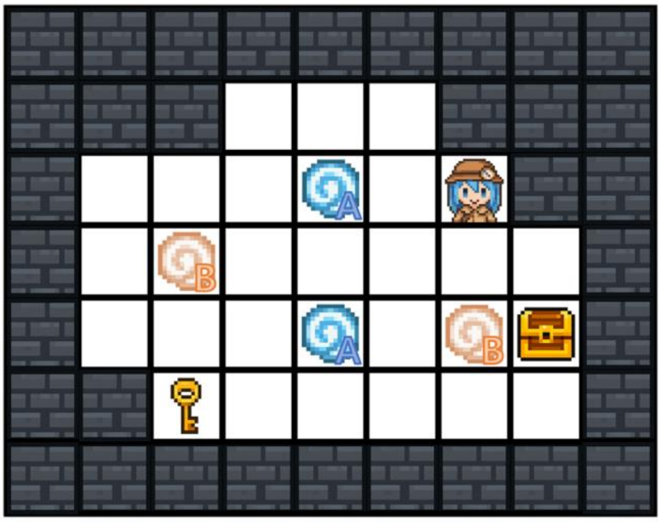


Figure 1. An example configuration of the underground city ($h = 7$; $w = 9$) with two teleport pairs `A` and `B`.

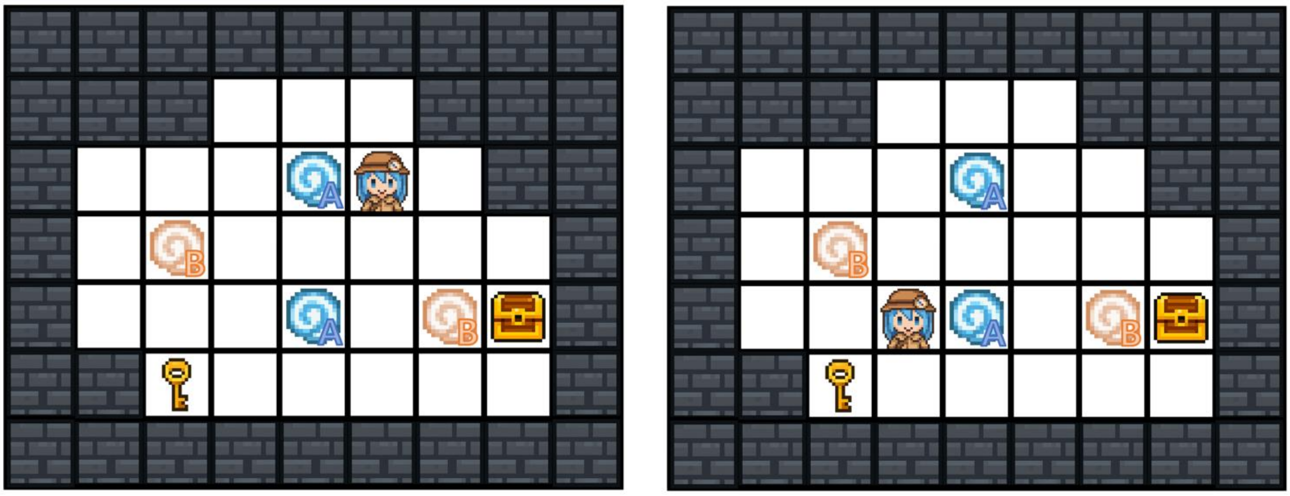
Dabi can perform the following two actions:

- Move to one of the **four** adjacent cells that is **not** a wall. Teleportation and the subsequent extra step are considered part of 'entering a teleport cell' and do not count as separate actions.
- Pick up the key if she is on the cell with the key.

Dabi always stands on an empty cell before and after each action.

Figure 1 illustrates an example of the underground city where $h = 7$ and $w = 9$. Teleport pair `A` connects the two cells marked with the letter `A`, and teleport pair `B` connects the two cells marked with the letter `B`. Note that Figure 1 corresponds to the city described in the attached `sample.in` file.

As shown in Figure 2, the teleportation process works as follows. In Figure 2(a), Dabi is standing on the cell immediately east of teleport `A`. When she moves west, she steps into the teleport cell marked `A`, is instantly transported to its paired cell, and then moves one additional step to the west, resulting in the configuration shown in Figure 2(b).



(a) **(b)**

Figure 2. Illustration of the teleportation process. **(a)** Dabi stands on the cell immediately east of teleport `A`. **(b)** After moving west, she steps into teleport cell `A`, is transported to its paired cell, and then moves one additional step to the west.

When Dabi picks up the key, the city begins to collapse. From that moment, she must reach the treasure cell in the **minimum** possible number of actions. Any longer route causes the attempt to fail.

Dabi carries a reliable compass and can always tell north, south, east, and west. By feeling the walls around her, she can sense for each of the **four** directions whether the adjacent cell is a **wall** or not. However, she does not initially know her coordinates or the overall layout of the city. At any moment, she can also sense whether her current cell contains the **key** or the **treasure**. She may stand on the key cell **without** immediately picking it up.

Your task is to guide Dabi through the city to obtain the key and then reach the treasure chest, obeying all movement rules. Write a program to accomplish this by interacting with the interactor through a sequence of commands.

Interaction

This is an **interactive** problem. Your submitted program will interact with an **interactor** inside the grading server, which reads input from and writes output to your program. Your program must control Dabi by printing commands and reading the interactor's replies. After every line of output, you must immediately **flush** the output.

Initial Input

At the beginning of the interaction, the interactor provides information about Dabi's current cell. The information is given as a single line containing six characters without spaces:

- Each of the six characters is either '`0`' or '`1`'.
- For the first four characters, '`1`' means that the adjacent cell in that direction is a wall, and '`0`' means that the adjacent cell in that direction is **not** a wall. The order of directions is north, south, east, and west.
- The fifth character is '`1`' if the key is present on Dabi's current cell and has not yet been picked up; otherwise, it is '`0`'.
- The sixth character is '`1`' if the treasure chest is on Dabi's current cell; otherwise, it is '`0`'.
- Since the key and the treasure chest are on different cells, the fifth and sixth characters are never both '`1`'.

For example, "`100010`" means that there is a wall to the north, the other three directions are open, the unpicked key is on the current cell, and there is no treasure.

Commands

Your program may repeatedly output exactly one of the following commands until the interactor outputs "`correct`" or terminates the interaction.

1. "`N`", "`S`", "`E`", or "`W`"
   - Move one step in the chosen direction—"`N`" (north), "`S`" (south), "`E`" (east), or "`W`" (west). The target cell must **not** be a wall. If it is a teleport cell, Dabi will be teleported according to the rules described above.
   - Attempting to move into a wall causes the interactor to output "`wrong`".
2. "`K`"
   - Pick up the key on the current cell. After this action, the treasure chest opens, and from that moment Dabi must reach the treasure cell in the **minimum** possible number of actions.
   - If there is no unpicked key on the current cell, the interactor outputs "`wrong`".

Each command must be printed on its own line and immediately flushed. The total number of commands issued **must not exceed** 230,611.

Interactor Responses

After each action, the interactor responds as follows:

- The interactor outputs "`correct`" if Dabi has already picked up the key and has just reached the treasure cell using the minimum possible number of actions.
- The interactor immediately terminates the interaction if any rule is violated—for example, a malformed command, an invalid move, attempting to pick up a nonexistent or already collected key, exceeding 230,611 actions in total, or reaching the treasure through a route that does not use the minimum possible number of actions after the key has been collected.
- Otherwise, the interactor outputs a six-character string describing Dabi's new current cell, in the same format as the initial input.

Receiving "`correct`" means that your program has successfully completed the mission and should terminate gracefully. **Do not forget to flush the output** after printing each command.

The city layout is fixed throughout the interaction; the interactor is **not adaptive**.

The time and memory used by the interactor are also included in the calculation of your program's execution time and memory usage. You can assume that the maximum time used by the interactor is 1 second and the maximum amount of memory is 64 MiB.

The following shows a sample interaction when the city layout and Dabi's initial position are the same as in Figure 1.

Sample Interaction 1

Read (Interactor's Response)	Write (Your Command)
`101000`	
	`W`
`000000`	
	`W`
`000000`	
	`S`
`010000`	
	`W`
`010110`	
	`K`
`010100`	
	`N`
`000000`	
	`W`
`010100`	
	`N`
`000100`	
	`E`
`correct`	

To flush, you need to do the following right after writing a command and a newline:

- `fflush(stdout)` in C;
- `std::cout << std::flush` in C++;
- `System.out.flush()` in Java or Kotlin;
- `sys.stdout.flush()` in Python.

A testing tool is provided to help you develop your solution, so it is not mandatory to use it. If you want to use it, you can download the attachment `testing_tool.py` from the *DOMjudge Problemset* page. You can run the testing tool to see how your program interacts for a specific city layout. **This testing tool can be used regardless of the language of your program as follows.** The same explanation is also included in the comments of `testing_tool.py`.

Usage: python3 testing_tool.py -f <inputfile> <program>

Use the `-f` parameter to specify the input file, e.g. `input.in`.

Format of the input file: The first line contains two integers h and w. The next h lines each contain a string of length w, describing the map of the city. Each character represents a cell type as follows:

- '`#`': A wall cell.
- '`.`': An empty cell.
- '`k`': The empty cell containing the key.
- '`t`': The empty cell containing the treasure chest.
- '`v`': The empty cell where Dabi initially stands.
- '`A`', '`B`', '`C`', '`D`', '`E`', '`F`': Teleport cells, where each letter represents one teleport pair.

Example: The city shown in Figure 1 is represented as follows.

```
7 9
#########
###...###
#...A.v##
#.B.....#
#...A.Bt#
##k.....#
#########
```

The tool is provided as-is, and you should feel free to make whatever alterations or augmentations you like to it. **Note that it is not guaranteed that a program that passes the testing tool will be accepted**. For example, the testing tool **does not verify** whether Dabi reaches the treasure chest using the minimum possible number of actions after picking up the key; instead, it only reports how many actions were taken.

If you have a C++ solution stored in a file called "`sol.cpp`", you must first compile using "`g++ sol.cpp -o sol`" and then invoke the testing tool with:

```
python3 testing_tool.py -f input.in ./sol
```

If you have a Python solution that you would run using "`pypy3 solution.py`", you can invoke the testing tool with:

```
python3 testing_tool.py -f input.in pypy3 solution.py
```

If you have a Java solution that you would run using "`java MyClass`", you can invoke the testing tool with:

```
python3 testing_tool.py -f input.in java MyClass
```

If you have a Kotlin solution that you would run using "`kotlin SolutionKt`", you can invoke the testing tool with:

```
python3 testing_tool.py -f input.in kotlin SolutionKt
```

Problem B
Badge Relay
Time Limit: 6 Seconds

A research facility operates two buildings, **Left Lab** and **Right Lab**, connected by a single secure corridor. At the beginning, all selected employees are standing in the Left Lab.

- The corridor can hold **at most two people at a time**.
- There is **exactly one security badge**, and every traversal of the corridor (whether by one person or two together) must be accompanied by the badge.
- If **two people travel together**, they must walk **side by side**, and the traversal time is equal to the **slower** person's time.
- If there are still employees remaining in the Left Lab after a traversal to the Right Lab, then **someone in Right Lab must bring the badge back** to the Left Lab before the next traversal from the Left Lab to the Right Lab can begin.

To see how the strategy affects the total time, consider four employees whose individual traversal times $\{1, 2, 5, 10\}$. One possible strategy is as follows. If $(1, 2)$ traverse first (taking 2 minutes), then (1) returns with the badge (1 minute), then $(1, 5)$ traverse (5 minutes), (1) returns (1 minute), and finally $(1, 10)$ traverse (10 minutes), the total time is **19 minutes**. However, there is another (better) strategy; if $(1, 2)$ traverse first (2 minutes), (1) returns (1 minute), then $(5, 10)$ traverse together (10 minutes), (2) returns (2 minutes), and finally $(1, 2)$ traverse again (2 minutes), the total time becomes **17 minutes**, which yields a smaller total traversal time than the first. For convenience, the two crossing sequences are summarized in Tables 1 and 2 below.

Table 1. Sequence A (Total 19 min)

Step	Action	Left Lab (after)	Right Lab (after)	Duration (min)	Elapsed (min)
0	-	{1,2,5,10}	∅	-	-
1	Cross (1, 2)	{5,10}	{1,2}	2	2
2	Return (1)	{1,5,10}	{2}	1	3
3	Cross (1, 5)	{10}	{1,2,5}	5	8
4	Return (1)	{1,10}	{2,5}	1	9
5	Cross (1, 10)	∅	{1,2,5,10}	10	19

Table 2. Sequence B (Total 17 min)

Step	Action	Left Lab (after)	Right Lab (after)	Duration (min)	Elapsed (min)
0	-	{1,2,5,10}	∅	-	-
1	Cross (1, 2)	{5,10}	{1,2}	2	2
2	Return (1)	{1,5,10}	{2}	1	3
3	Cross (5, 10)	{1}	{2,5,10}	10	13
4	Return (2)	{1,2}	{5,10}	2	15
5	Cross (1,2)	∅	{1,2,5,10}	2	17

You are given n employees. Employee i needs T_i minutes to traverse the corridor. You are also given q queries. Each query is specified by:

- an index range $[x, y]$ (inclusive),
- a time range $[a, b]$ (inclusive), and
- a selection cap K (the maximum number of employees you may select).

For each query, consider all employees whose indices lie in $[x, y]$ and whose traversal times lie in $[a, b]$. Among these employees, select the K employees who have the smallest individual traversal times. If fewer than K such employees exist, select all of them. These selected employees all start in the **Left Lab** with the single badge. Under the rules described above, you should compute the **minimum total traversal time** required for **all selected employees** to reach the **Right Lab**. If no employee is selected, the answer should be zero.

Given n employees with traversal times $T_1, \cdots, T_n$ and q queries of the form (x, y, a, b, K), write a program that processes the queries and outputs, for each query, the minimum total traversal time for the selected employees to reach the Right Lab.

Input

Your program is to read from standard input. The input starts with a line containing two integers n and q $(1 \le n, q \le 100{,}000)$, where n is the number of employees and q is the number of queries. Employees are numbered from 1 to n. The second line contains n integers $T_1, \cdots, T_n$ $(1 \le T_i \le 10^9)$, where T_i is the traversal time of employee i $(1 \le i \le n)$. Each of the following q lines contains five integers x, y, a, b, K described above, where $1 \le x \le y \le n$; $1 \le a \le b \le 10^9$; $1 \le K \le n$.

Output

Your program is to write to standard output. For each query, print a single line containing the minimum total traversal time for the selected employees to move from the Left Lab to the Right Lab. If no employee is selected, output 0.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
3 3 1 2 3 1 3 1 3 3 1 3 1 3 2 1 3 4 5 1	6 2 0

Sample Input 2	Output for the Sample Input 2
4 4 5 1 10 2 1 4 1 10 4 1 4 2 10 2 1 4 2 10 4 1 3 1 13 3	17 5 17 16

Problem C
Bay
Time Limit: 1 Second

We have a grid (lattice) graph $G(n, n)$, where n is the number of vertices along both the x-axis and the y-axis, that is, the number of rows and columns. The vertices of the graph $G(n, n)$ are numbered consecutively from 1 to n^2 in row-major ordering; starting from the top-left vertex, we traverse row by row from top to bottom, and within each row from left to right. Figure 1 shows two examples, $G(5, 5)$ and $G(7, 7)$ with vertex numbers.

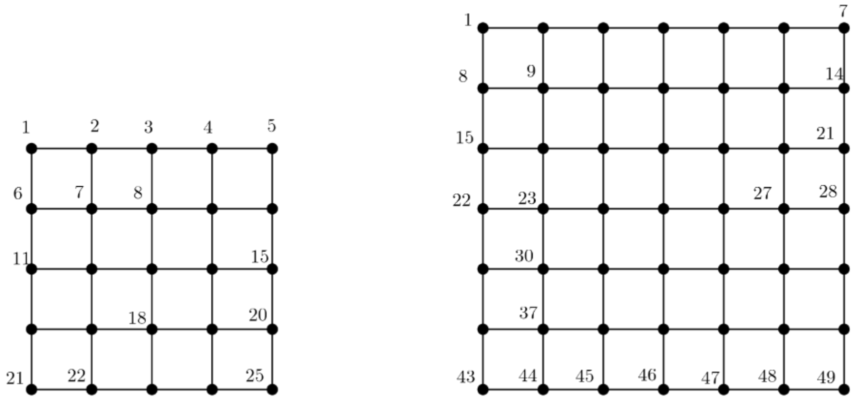


Figure 1. Left: Grid graph $\boldsymbol{G(5,5)}$. Right: $\boldsymbol{G(7,7)}$.

We are given a spanning tree T of $G(n, n)$. The left in Figure 2 shows a spanning tree T of $G(7, 7)$. If we add an edge of $G(n, n)$ that does not belong to T (called *non-tree edge*), then exactly one simple cycle is created. We define the region enclosed by this cycle as a *bay*. There is a one-to-one correspondence between non-tree edges and bays, that is, each non-tree edge corresponds to exactly one bay. The area of a bay is defined by number of 1×1 unit cells enclosed by the cycle. The right in Figure 2 shows two bays (colored blue and orange) created by adding two non-tree edges (u, v) and (p, q), respectively. Note that the areas of two bays created by (u, v) and (p, q) are 4 and 12, respectively.

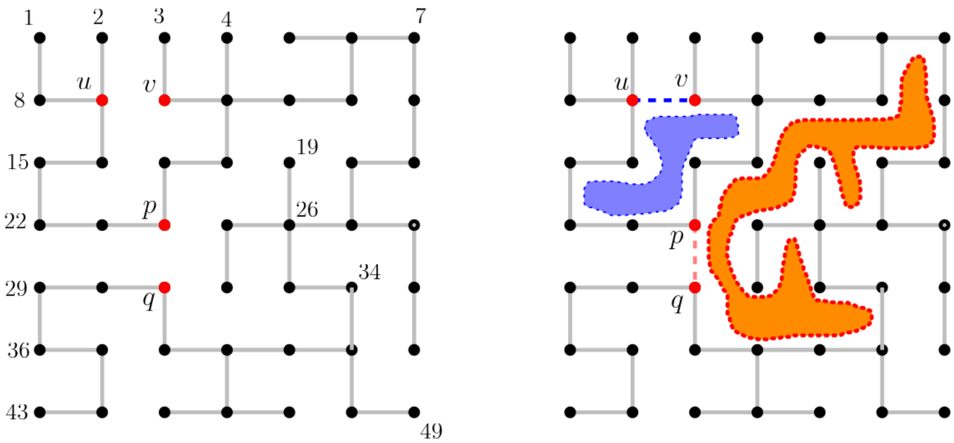


Figure 2. A spanning tree $\boldsymbol{T}$ of a grid $\boldsymbol{G(7,7)}$ and two bays created by $\boldsymbol{(u,v)}$ and $\boldsymbol{(p,q)}$.

Given a spanning tree T of a grid graph $G(n, n)$ and a positive integer S, write a program that finds all non-tree edges that creates bays of area S and outputs the first non-tree edge among them in lexicographical order.

Input

Your program is to read from standard input. The input starts with a line containing two integers n and S where $5 \leq n \leq 300$ for $G(n, n)$ and $1 \leq S \leq (n-1)^2$. Each of the following $n^2 - 1$ lines contains two distinct integers u and v representing an edge (u, v) of a spanning tree T, where $1 \leq u < v \leq n^2$.

Output

Your program is to write to standard output. The first line should contain the number of non-tree edges that create the bays of area S. The second line should contain two distinct integers u and v $(u < v)$ representing the first non-tree edge (u, v) in lexicographical order among those that create the bays of area S. The lexicographical order of two edges (a, b) and (c, d) is defined such that (a, b) comes before (c, d) if and only if $a < c$ or, if $a = c$ then $b < d$. If there is no non-tree edge that creates the bay of area S, then print “`0`” in the first line and two zeros “`0 0`” in the second line.

Figure 3 shows two spanning trees of a grid graph with $n = 5$, which are the sample inputs and outputs.

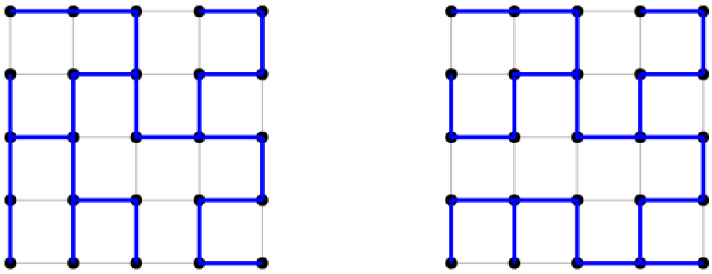
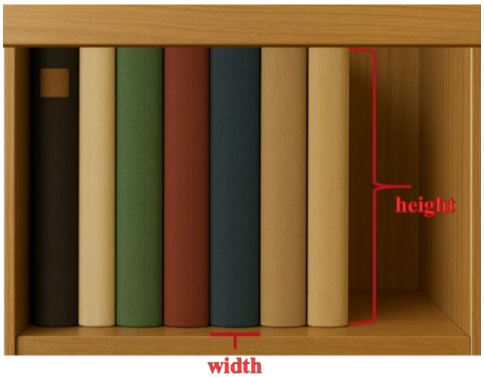


Figure 3. Two spanning trees of $G(5, 5)$ for Sample Input 1 and 2.

Sample Input 1	Output for the Sample Input 1	Sample Input 2	Output for the Sample Input 2
5 2	2	5 2	0
1 2	13 18	1 2	0 0
2 3		2 3	
3 8		3 8	
4 5		4 5	
5 10		5 10	
6 11		6 11	
7 8		7 8	
7 12		7 12	
8 13		8 13	
9 10		9 10	
9 14		9 14	
11 12		11 12	
11 16		13 14	
12 17		14 15	
13 14		15 20	
14 15		16 17	
15 20		16 21	
16 21		17 18	
17 18		17 22	
17 22		18 23	
18 23		19 20	
19 20		19 24	
19 24		23 24	
24 25		24 25	

Problem D
Bookshelf
Time Limit: 1 Second



A bookshelf of length L holds n books, $B_1, \cdots, B_n$, arranged from left to right. Each book B_i has a width (thickness) of w_i. The heights of the bookshelf and the books are the same. Position x on the shelf corresponds to a point located x units far from the left end. If a book B_i is placed at position x, it occupies the interval $[x, x + w_i)$ on the shelf. Then the intervals of the books on the shelf are pairwise disjoint. The left end of the shelf is at position 0, the right end is at position L, and the shelf as a whole occupies the interval $[0, L)$.

Rearranging the books currently on the shelf, you may perform the following operation any number of times:

- Choose one book B_i on the shelf and take it out, which creates a contiguous empty interval where it was.
- Then insert B_i into any existing empty interval on the shelf whose length is at least w_i.

During this operation, all other books that remain on the shelf stay fixed—cannot slide, move, or be nudged in any way. This is because the books and the shelf have the same height and fit tightly together, so no book can move unless it is explicitly taken out. Also, you are not allowed to push or shift any other books to make room during the operation.

The owner has a favorite book B_k among n books on the shelf and wishes to place it at a specific position p.

Given the initial positions of the books on the shelf, the favorite book B_k, and its target position p, determine whether it is possible to place B_k at position p after performing any number of the above operations—possibly zero.

Input
Your program is to read from standard input. The input starts with a line containing two integers n and L ($1 \le n \le 100{,}000$; $1 \le L \le 10^9$), where n is the number of books and L is the length of shelf. The second line contains n distinct integers between 0 and $L - 1$ (inclusive), representing the positions of books $B_1, \cdots, B_n$ initially arranged on the shelf in ascending order. The third line contains n positive integers, where the i-th integer ($1 \le i \le n$) is the width w_i of the i-th book B_i in the initial arrangement. The next line contains two integers k and p ($1 \le k \le n$; $0 \le p \le L - 1$), where the k-th book B_k in the initial arrangement is the favorite one and its target position is p.

Output
Your program is to write to standard output. Print exactly one line. Print "`YES`" if it is possible to place the favorite book at the target position, and print "`NO`" otherwise.

The following shows sample input and output for four test cases.

Sample Input 1

```
3 6
1 3 5
1 2 1
3 3
```

Output for the Sample Input 1

```
YES
```

Sample Input 2

```
3 6
1 3 5
1 2 1
2 5
```

Output for the Sample Input 2

```
NO
```

Sample Input 3

```
3 7
0 3 6
2 3 1
3 1
```

Output for the Sample Input 3

```
YES
```

Sample Input 4

```
3 7
0 3 6
2 3 1
3 4
```

Output for the Sample Input 4

```
NO
```

Problem E

Clean Arrangements

Time Limit: 1 Second

In graph drawing, a *linear arrangement* of a rooted (connected) tree $T = (V, E)$ of n vertices is a planar drawing where n vertices of the tree are placed on a horizontal line, say the x-axis, and $(n-1)$ edges are drawn as semicircular arcs above the line connecting their end vertices as shown in Figure 1. Such linear arrangement π maps each vertex to a distinct integer from 1 to n, representing its coordinate along the x-axis.

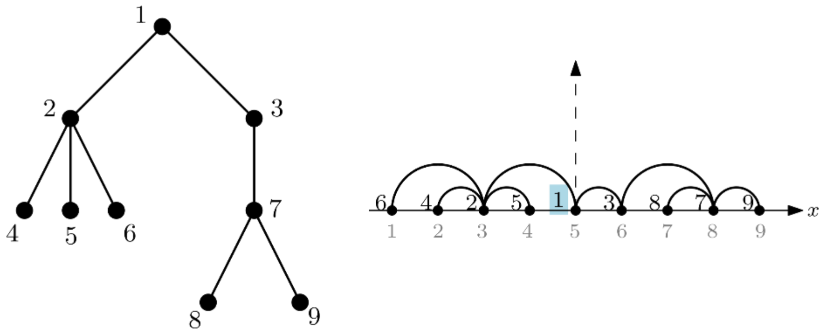


Figure 1. (Left) A rooted tree T of nine vertices with the vertex 1 as a root.
(Middle) A clean arrangement of T.
(Right) An unclean arrangement of T because of the red edge $(3, 7)$ covering the root.

In a linear arrangement π, the distance $d(u, v)$ between two vertices u and v is defined as the difference of their x-coordinates, i.e., $d(u, v) = |\pi(u) - \pi(v)|$. Formally, for a rooted tree $T = (V, E)$, the cost of a linear arrangement π of T is defined as $\sum_{(u,v)\in E} d(u, v)$.

A *clean arrangement* π of a rooted tree T is a special linear arrangement π satisfying both conditions:

1. π has no edge crossings except at common end vertices of edges.
2. No edge covers the root vertex r of T, that is, there is no edge (u, v) such that $\pi(u) < \pi(r) < \pi(v)$.

For example, the middle in Figure 1 is a clean arrangement of T in the left, but the right is not clean because the edge $(3, 7)$ covers the root vertex 1. The cost of the clean arrangement in the middle is 11, where there are three edges of distance two and five edges of distance one. This cost is the minimum among all clean arrangements of T.

Given a rooted tree with the vertex 1 as a root, write a program to output the minimum possible cost of clean arrangements of the tree.

Input

Your program is to read from standard input. The input starts with a line containing n $(2 \leq n \leq 5{,}000)$, where n is the number of vertices of the rooted tree. The vertices are numbered from 1 to n, and the root vertex is 1. In the following $(n-1)$ lines, each line contains two positive integers u and v which are end vertices of an (undirected) edge (u, v) of the tree, where u and v are distinct integers between 1 and n.

Output

Your program is to write to standard output. Print exactly one line. The line should contain the minimum cost of clean arrangements of the tree with root vertex 1.

The following shows sample input and output for two test cases. The second test case corresponds to Figure 1.

Sample Input 1	Output for the Sample Input 1
4 1 2 3 2 2 4	4

Sample Input 2	Output for the Sample Input 2
9 2 4 2 5 7 3 9 7 1 2 3 1 7 8 6 2	11

Problem F
CPEquivalence
Time Limit: 2 Seconds

Given an integer array x consisting of integers (namely, each item of x is an integer), the Closest Position array (CP-array) $\mathrm{CP}(x)$ is an array of length $|x|$ defined to be

$$\mathrm{CP}(x)[i] = \max(\{j \mid j < i;\ x[j] \geq x[i]\} \cup \{-1\}) \text{ for all } 0 \leq i < |x|,$$

where $x[i]$ denotes the i-th integer in x, and the length $|x|$ is the number of integers in x. In other words, $\mathrm{CP}(x)[i]$ is the greatest index of x that is smaller than i and whose item at that index is greater than or equal to $x[i]$. For example, when $x = [64, 2, 5, 100, 100]$, its CP-array is $\mathrm{CP}(x) = [-1, 0, 0, -1, 3]$, and $|x| = 5$.

We say that two integer arrays x and y are CP-equivalent if $\mathrm{CP}(x) = \mathrm{CP}(y)$. It is obvious that two CP-equivalent integer arrays x and y have the same length. For example, two arrays $x = [64, 2, 5, 100, 100]$ and $y = [3, 1, 2, 4, 1]$ are CP-equivalent because their CP-arrays are the same as $[-1, 0, 0, -1, 3]$.

For an integer array x, an integer a and a non-negative integer $i < |x|$, a substitution operation on x at position i into a returns the array $\left[x[0], x[1], \cdots, x[i-1], a, x[i+1], \cdots, x[|x|-1]\right]$. For an integer array x and a non-negative integer $i < |x|$, a deletion operation on x at position i returns the array $\left[x[0], x[1], \cdots, x[i-1], x[i+1], \cdots, x[|x|-1]\right]$. Finally, for an integer array x, an integer a and a non-negative integer $i \leq |x|$, an insertion operation on x at position i returns the array $\left[x[0], x[1], \cdots, x[i-1], a, x[i], \cdots, x[|x|-1]\right]$. An edit operation on x is one of an insertion, a deletion or a substitution at a single position.

Given two integer arrays x and y, compute the minimum number of edit operations on y to obtain an array y' satisfying $\mathrm{CP}(x) = \mathrm{CP}(y')$.

For example, let $x = [64, 2, 5, 100, 100]$ and $y = [-5, -5, -5, -5]$. Consider the array $y' = [-5, -6, -5, -4, -5]$. Then, we have $\mathrm{CP}(y') = [-1, 0, 0, -1, 3]$ and therefore x and y' are CP-equivalent. Then, we can obtain the integer array y' applying two edit operations on y and it is minimum.

Input
Your program is to read from standard input. The input consists of three lines. The first line consists of two integers n and m ($1 \leq n \leq 40$; $1 \leq m \leq 40$) that indicate the length of x and y, respectively. The second line consists of n integers between $-1{,}000{,}000$ and $1{,}000{,}000$ (both inclusive), representing the array x. The third line consists of m integers between $-1{,}000{,}000$ and $1{,}000{,}000$ (both inclusive), representing the array y.

Output
Your program is to write to standard output. Print exactly one line containing the minimum number of edit operations on y to obtain an integer array y' satisfying $\mathrm{CP}(x) = \mathrm{CP}(y')$.

The following shows sample input and output for four test cases.

Sample Input 1

```
5 5
64 2 5 100 100
3 1 2 4 1
```

Output for the Sample Input 1

```
0
```

Sample Input 2

```
5 4
64 2 5 100 100
-5 -5 -5 -5
```

Output for the Sample Input 2

```
2
```

Sample Input 3

```
6 5
1 2 3 4 5 6
2 5 3 4 5
```

Output for the Sample Input 3

```
3
```

Sample Input 4

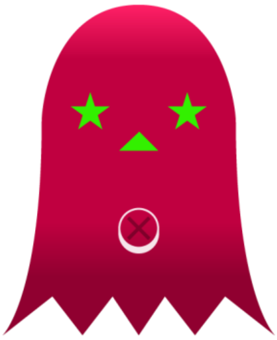
```
6 3
1 3 5 2 5 2
5 5 6
```

Output for the Sample Input 4

```
3
```

Problem G
Extraterrestrial Creatures
Time Limit: 2 Seconds

In the year 3025, ICPC (Interplanetary Constitution for Peculiar Creatures) found an exotic animal in asteroid KP-124. Upon further inspection, ICPC succeeded in figuring out how they live and how their ecosystem works altogether:



- They have a button on their belly, shaped just like a belly button of us earthers.
- On their head is a series of strange-looking symbols that works just like the decimal system of us earthers. ICPC already has a knowledge of what each of the symbols means, so for you as an earther we will just use the term "their number" and the standard decimal notation to represent the values.
- When their button is pressed, their number increases by a fixed value, possibly different for each entity. They try to hit their button as much as they can since each button press increases their chance of survival.

The researchers on KP-124 quickly became fascinated by these creatures and kept n of those as pets in the research station to amuse themselves from time to time. Let us give them a unique id from 1 to n. The mission on KP-124 was a success with the pets' emotional support and it was time for the researchers to leave the asteroid. As a farewell present to the pets, you, one of the researchers, decided to press buttons a total of X times. To ensure an even chance of survival among the creatures, you made a rule to press the button on one that has the smallest number on its head each time. If there is a tie, you choose the one having the smallest id among those tied.

Creature id	Initial number	Increment
1	5	3
2	1	4
3	3	6

For example, let $n = 3$, $X = 3$, and the information of the 3 pets be as the table on the right. Initially they have the numbers $[5, 1, 3]$. On the first press, you will press the button on creature 2, since it has the smallest number. Now the numbers become $[5, 5, 3]$, so that the smallest will be creature 3 and you will press its button. Then the numbers become $[5, 5, 9]$ where the smallest one is tied between creature 1 and 2. Since creature 1 has the smallest id, you will press the button on creature 1, making the numbers on them $[8, 5, 9]$.

Given the information about the creatures before pressing their buttons, write a program to find the resulting numbers on the creatures' heads.

Input

Your program is to read from standard input. The input starts with a line containing two integers, n and X ($1 \le n \le 500{,}000$; $1 \le X \le 10^{12}$), where n and X are as explained above. The second line contains n nonnegative integers, i-th of which is the number initially written on the head of creature i. The third line contains n positive integers, i-th of which is how much the value on creature i is increased by when its button is pressed. All the integers on the second and the third lines are no more than 10^6.

Output

Your program is to write to standard output. Print exactly one line. The line should contain n integers, i-th of which is the number written on the head of creature i after buttons are pressed X times in total.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
3 3 11 1 22 14 5 1	25 11 22

Sample Input 2	Output for the Sample Input 2
9 5 9 8 6 2 1 6 5 10 9 9 3 9 1 1 4 10 5 3	9 8 6 4 4 6 5 10 9

Problem H
Fair Problemset
Time Limit: 8 Seconds

This problem adopts exactly the same definition of *Fair Problemset* as Problem M, "Triple Fairness".

ICPC is a team competition. Each team has three members. At the beginning of a contest, most teams divide the $3n$ problem evenly. They use one of two common methods to distribute problems:

1. **Sequential Distribution**: Each member takes a contiguous block of n problems from the set of $3n$ problems. Specifically, the first member takes problems $1, \cdots, n$, the second member takes problems $n+1, \cdots, 2n$, and the third member takes problems $2n+1, \cdots, 3n$.
2. **Jump Distribution**: Each member takes problems with indices that have the same remainder when divided by 3 from the set of $3n$ problems. Specifically, the first member takes problems $1, 4, 7, \cdots, 3n-2$, the second member takes problems $2, 5, 8, \cdots, 3n-1$, and the third member takes problems $3, 6, 9, \cdots, 3n$.

The ICPC Seoul Regional Contest Scientific Committee must prepare a problemset consisting of $3n$ problems. The difficulty of each problem is represented by an integer from 1 to n, inclusive. For each difficulty, there are exactly three problems with that difficulty. Thus, the arrangement of difficulties in the problemset can be viewed as a *difficulty sequence* of length $3n$ containing three problems of each of the n difficulty levels.

To prevent any advantage or disadvantage for a team based on their chosen problem distribution method, the ICPC Seoul Regional Contest Scientific Committee has defined a standard called a *Fair Problemset*. A difficulty sequence of length $3n$ is called a Fair Problemset if it satisfies both of the following conditions:

1. **Sequential Distribution Fairness**: When using Sequential Distribution, for every difficulty level i $(1 \leq i \leq n)$, each of the three members receives exactly one problem with difficulty i.
2. **Jump Distribution Fairness**: When using Jump Distribution, for every difficulty level i $(1 \leq i \leq n)$, each of the three members receives exactly one problem with difficulty i.

In other words, regardless of which of the two methods is chosen, each team member must be assigned exactly one problem for each difficulty level from 1 to n, inclusive.

Given a positive integer k, write a program to find the number of Fair Problemset sequences of length $3n$ for each $n = 1, 2, \cdots, k$.

Input
Your program is to read from standard input. The input consists of exactly one line. The line contains two integers, k and m ($1 \leq k \leq 10^6$; $10^8 < m < 10^9$; m is a prime number).

Output
Your program is to write to standard output. You should print exactly k lines. On the n-th line ($1 \leq n \leq k$), print the number of Fair Problemset sequences of length $3n$, modulo m.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
2 993244853	1 2

Sample Input 2	Output for the Sample Input 2
3 998244353	1 2 12

Explanation of Sample Input 2:

Here are 12 Fair Problemset sequences of length 9 $(= 3 \times 3)$.

i. 1 2 3 2 3 1 3 1 2
ii. 1 2 3 3 1 2 2 3 1
iii. 1 3 2 2 1 3 3 2 1
iv. 1 3 2 3 2 1 2 1 3
v. 2 1 3 1 3 2 3 2 1
vi. 2 1 3 3 2 1 1 3 2
vii. 2 3 1 1 2 3 3 1 2
viii. 2 3 1 3 1 2 1 2 3
ix. 3 1 2 1 2 3 2 3 1
x. 3 1 2 2 3 1 1 2 3
xi. 3 2 1 1 3 2 2 1 3
xii. 3 2 1 2 1 3 1 3 2

Problem I
Magic Door
Time Limit: 0.5 Seconds

The magic door to a treasure cave is an $m \times n$ grid filled with various gems. The gems fall into three categories: **regular gems** of various colors and shapes, and two special types - **bomb gems** and **vibranium gems**.

A *match-3 group* is a group of three or more identical regular gems aligned consecutively in a vertical or horizontal direction. A single gem may belong to both a horizontal match-3 group and a vertical match-3 group at which they intersect each other. Bomb gems do not form match-3 groups. Vibranium gems do not either.

In the initial state of the grid, no match-3 groups exist. When you initially touch two regular gems with your hand, they swap their positions. This single swap triggers a chain reaction that causes the gems to disappear from the grid according to a set of specific rules.

After the initial swap, the following process repeats until no more changes occur on the grid:

- **Stage 1: Chain Reactions of Match-3 Groups**
    1. All gems belong to every match-3 group disappear simultaneously.
    2. All types of gems above the vanished gems fall downward due to gravity, filling the empty spaces. During this process, any bomb gem that moves down by at least one cell becomes *activated*.
    3. If new match-3 groups are formed after the gems fall, this stage repeats from the first sub-step.
    4. If no more match-3 groups are formed, proceed to Stage 2.

- **Stage 2: Activated Bomb Detonation**
    1. All bombs currently in the *activated* state explode simultaneously.
    2. Each exploding bomb emits beams in horizontal and vertical directions.
    3. The beams travel from the bomb's position to the grid edge.
    4. A beam's path is blocked if it hits a vibranium gem.
    5. All regular gems and bombs on the beam's path disappear along with the exploded bomb.
    6. Unexploded bombs hit by a beam simply disappear without exploding.
    7. The gems above the vanished gems fall downward due to gravity, filling the newly created empty spaces. During this process, any bomb that moves down by at least one cell becomes *activated*. Then Stage 1 repeats.

Once no more changes occur due to Stage 1 and Stage 2, the entire process is terminated. Note that bomb gems and vibranium gems are not selected from the initial swap and do not form match-3 groups. Also note that any vibranium gem never disappears, acts as a barrier blocking the bomb's beam, and it is affected by gravity and will fall into empty spaces below it.

For example, consider a magic door composed of a 6×4 grid, as shown in Figure 1. In the figure, `0` represents a bomb gem, `-1` represents a vibranium gem, and positive numbers represent regular gems. Starting from the initial state shown in the figure (a), swapping the two gems located at $(6, 1)$ and $(5, 4)$ results in the state shown in (b). The location (r, c) denotes the cell in the r-th row from the top and the c-th column from the left. Coordinates are 1-based. This state contains a match-3 group consisting of three `2`s. According to Stage 1, when the gems in this group disappear, state (c) is reached. At this point, the bomb gem at $(4, 3)$ is activated and a new match-3 group consisting of three `1`s is formed. Again, Stage 1 removes the three gems of 1, resulting in state (d). Since no match-3 groups remain, the bomb explodes due to Stage 2. This results in state (e). Here, two match-3 groups consisting of three `1`s and four `3`s are formed. Again, when these gems disappear due to Stage 1, state (f) is reached. No further changes occur in this state, making it the final state. A total of 18 gems disappear during this process.

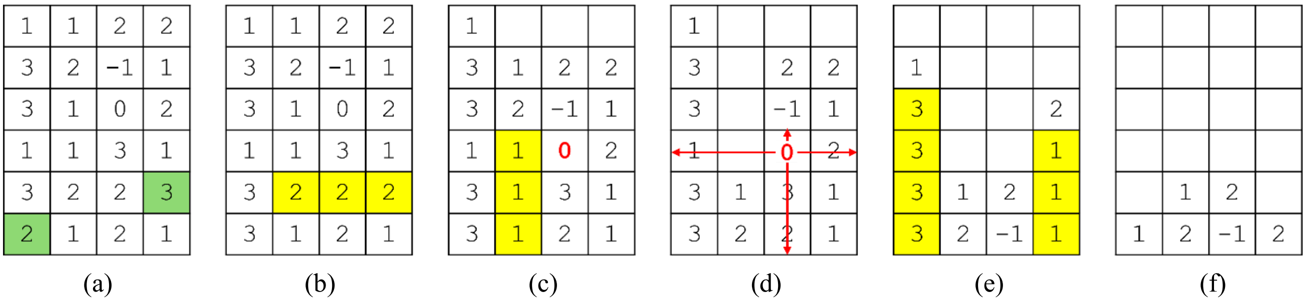


Figure 1. Chain Reactions

The magic door opens when a specific number of gems disappear. Your task is to write a program that, given the information about the magic door and the positions of the two regular gems to be swapped, calculates the number of gems that disappeared after the entire chain reaction, initiated by a single swap, comes to a complete stop.

Input

Your program is to read from standard input. The input starts with a line containing two integers, m and n ($3 \le m \le 80$; $3 \le n \le 80$), representing the number of rows and columns of the grid. The next m lines each contain n integers, representing the gems in each row of the grid, where `0` represents a bomb gem, `-1` represents a vibranium gem, and integers between `1` and `9` inclusive represent regular gems. The final line contains four integers, r_1, c_1, r_2, c_2 ($1 \le r_1, r_2 \le m$; $1 \le c_1, c_2 \le n$), where (r_1, c_1) and (r_2, c_2) represent the coordinates of the two distinct regular gems to be swapped. The coordinates are 1-based.

Output

Your program is to write to standard output. Print exactly one line. The line should contain an integer representing the total number of gems that will disappear after all processes have been completed.

The following shows sample input and output for two test cases. (Sample Input 1 corresponds to Figure 1(a).)

Sample Input 1	Output for the Sample Input 1
6 4 1 1 2 2 3 2 -1 1 3 1 0 2 1 1 3 1 3 2 2 3 2 1 2 1 6 1 5 4	18

Sample Input 2	Output for the Sample Input 2
4 5 0 2 1 2 2 0 1 2 2 0 1 1 2 1 0 2 -1 -1 -1 2 3 3 3 4	15

Problem J
Mex Culpa
Time Limit: 5 Seconds

The mex (shorthand for **m**inimum **ex**cluded value) of a sequence is the smallest non-negative integer that is not in the sequence. For example:

- $\operatorname{mex}(\{\ \}) = 0$
- $\operatorname{mex}(\{1, 2, 3\}) = 0$
- $\operatorname{mex}(\{5, 0, 1, 1, 4\}) = 2$
- $\operatorname{mex}(\{0, 5, 2, 1, 5, 0, 1, 2\}) = 3$

While the mex function has applications in combinatorial game theory, it is still a rather niche method for mapping a sequence to an integer. In the absence of a more organic problem, we have repurposed this concept to construct a task of a somewhat artificial nature. Sorry!

Write a program that, given two sequences of positive integers $a = [a_1, a_2, \cdots, a_n]$ and $b = [b_1, b_2, \cdots, b_n]$, evaluates the following recurrence: for $1 \le i \le n$,

$$f_i = \operatorname{mex}\left(\{f_j \mid 1 \le j \le i-1;\ a_i \le a_j + b_j;\ a_j \le a_i + b_i\}\right)$$

Input
Your program is to read from standard input. The first line contains a single integer, n ($1 \le n \le 250{,}000$), representing the length of the sequences. The second line contains n positive integers $a_1, a_2, \cdots, a_n$ ($1 \le a_i \le 10^9$) representing the sequence a. The third line contains n positive integers $b_1, b_2, \cdots, b_n$ ($1 \le b_i \le 10^9$), representing the sequence b.

Output
Your program is to write to standard output. Print exactly one line consisting of n space-separated integers, denoting $f_1, f_2, \cdots, f_n$.

The following shows sample input and output for three test cases.

Sample Input 1	Output for the Sample Input 1
3 3 1 5 2 2 4	0 1 1

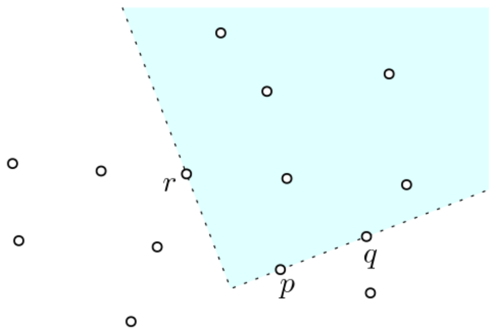
Sample Input 2	Output for the Sample Input 2
8 1 2 9 4 6 9 7 10 9 3 7 1 1 7 1 1	0 1 1 2 1 2 2 3

Sample Input 3	Output for the Sample Input 3
15 1 1 5 1 2 3 8 8 6 5 9 1 1 4 3 2 5 7 4 6 4 1 3 4 8 3 4 2 10 1	0 1 0 2 3 4 1 2 5 6 3 5 6 7 8

Problem K
Quadrants
Time Limit: 2 Seconds

This problem is about quadrants. What are quadrants? Let us begin with any two perpendicular lines ℓ and ℓ' in the plane $\mathbb{R}^2$. If you subtract the two lines ℓ and ℓ' from the whole plane $\mathbb{R}^2$, you obtain four connected, unbounded regions. Each of the four regions is called a *quadrant*. Note that the boundary of a quadrant does not belong to itself.

Now, consider a set P of points in the plane $\mathbb{R}^2$. We are interested in quadrants defined by the set P of points. Specifically, let $\mathfrak{Q}$ be the set of quadrants Q such that the boundary of Q contains exactly three points of P. Each quadrant $Q \in \mathfrak{Q}$ is called a k-*quadrant* if Q contains exactly k points of P in its interior. The figure below shows an example in which the set P consists of 14 points (small circles) and you can see a 5-quadrant $Q \in \mathfrak{Q}$ (shaded in cyan), whose boundary contains three points $p, q, r \in P$.



Given a set P of n points as input, write a program that computes the number of k-quadrants for every $0 \le k \le n - 3$.

Input
Your program is to read from standard input. The input starts with a line containing a single integer n ($3 \le n \le 2{,}000$), where n is the number of points in the input set P. In each of the following n lines, given are two integers x and y, both ranging from -10^6 to 10^6, inclusively, that represent the x- and y-coordinates of an input point (x, y) in P. You may assume that no two input points have the same coordinates, that there are no three points in P lying in a line, and that there are no two perpendicular lines ℓ and ℓ' in the plane $\mathbb{R}^2$ such that $|\ell \cap P| \ge 2$ and $|\ell' \cap P| \ge 2$.

Output
Your program is to write to standard output. Print exactly $n - 2$ lines. The i-th line of your output for each $1 \le i \le n - 2$ must contain a single integer that represents the number of $(i - 1)$-quadrants with respect to the input set P of n points.

The following shows sample input and output for three test cases.

Sample Input 1

```
3
0 0
1 2
-1 4
```

Output for the Sample Input 1

```
2
```

Sample Input 2

```
4
0 0
1 2
-1 4
-1 1
```

Output for the Sample Input 2

```
0
4
```

Sample Input 3

```
10
47 20
4 30
3 21
44 12
46 34
18 18
19 50
48 23
22 3
19 22
```

Output for the Sample Input 3

```
2
12
20
18
20
30
28
16
```

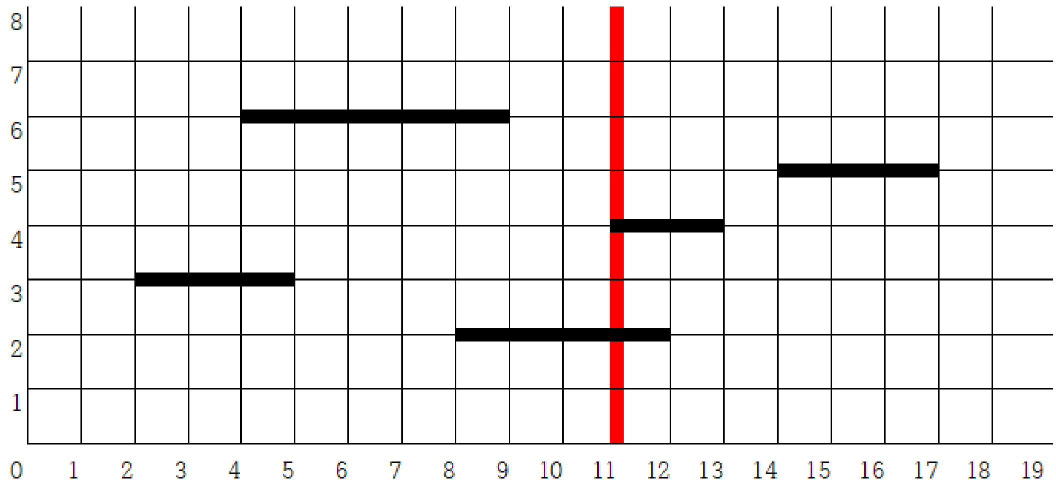
Problem L
Segments
Time Limit: 5 Seconds

In the first quadrant of a coordinate plane, you are given n line segments parallel to the x-axis. Each segment S_i $(1 \le i \le n)$ is represented by the coordinates of its left and right endpoints, (l_i, y_i) and (r_i, y_i), respectively. All coordinates are positive integers.

You must now answer q queries. For each query, a vertical line $x = p$, parallel to the y-axis, is given. The vertical line is represented by a single positive integer p.

If each segment S_i is horizontally extended, it will eventually meet the line $x = p$ at the point (p, y_i). If the segment, including its endpoints, already meets $x = p$, no extension is needed. For example, suppose there are 5 segments $\{(2,3),(5,3)\}$, $\{(4,6),(9,6)\}$, $\{(8,2),(12,2)\}$, $\{(11,4),(13,4)\}$, and $\{(14,5),(17,5)\}$, and a single line $x = 11$. The first segment must be extended by 6 to the right, the second segment 2 to the right, the third and the fourth segments 0, and the fifth segment 3 to the left for each to meet $x = 11$.

For each query, determine the maximum among the extension lengths required for all segments to meet the line $x = p$. Formally, let $\text{dist}(p, S_i)$ denote the distance that segment S_i must be extended to intersect $x = p$ at (p, y_i). For each query, output $\max_{1 \le i \le n} \text{dist}(p, S_i)$. In the example above, the answer to the query is 6. See the figure below.



Given n segments and q queries, write a program to output the maximum extension length for each query.

Input

Your program is to read from standard input. The input starts with a line containing two integers n ($1 \le n \le 2 \times 10^6$) and q ($1 \le q \le 2 \times 10^6$), where n is the number of line segments and q is the number of queries. In the following n lines, the i-th line contains three integers, l_i, r_i, and y_i ($1 \le l_i \le r_i \le 10^9$; $1 \le y_i \le 10^3$), where l_i (resp. r_i) is the x-coordinate of left (resp. right) endpoint of S_i and y_i is the y-coordinate of both endpoints of S_i. In the following q lines of queries, the j-th line contains one integer p_j $\left(1 \le p_j \le 10^9\right)$ which denotes the vertical line $x = p_j$.

Output

Your program is to write to standard output. Print exactly one line per each query. The j-th line should contain the maximum among the extension lengths required for all segments to meet $x = p_j$ at $\left(p_j, y_j\right)$.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
5 3 2 5 3 4 9 6 8 12 2 11 13 4 14 17 5 11 5 1	6 9 13

Sample Input 2	Output for the Sample Input 2
4 8 1 4 7 3 7 5 10 13 8 12 15 2 13 7 4 8 3 11 1 16	9 5 8 4 9 7 11 12

Problem M

Triple Fairness

Time Limit: 1 Second

This problem adopts exactly the same definition of *Fair Problemset* as Problem H, "Fair Problemset".

ICPC is a team competition. Each team has three members. At the beginning of a contest, most teams divide the $3n$ problem evenly. They use one of two common methods to distribute problems:

1. **Sequential Distribution**: Each member takes a contiguous block of n problems from the set of $3n$ problems. Specifically, the first member takes problems $1, \cdots, n$, the second member takes problems $n+1, \cdots, 2n$, and the third member takes problems $2n+1, \cdots, 3n$.
2. **Jump Distribution**: Each member takes problems with indices that have the same remainder when divided by 3 from the set of $3n$ problems. Specifically, the first member takes problems $1, 4, 7, \cdots, 3n-2$, the second member takes problems $2, 5, 8, \cdots, 3n-1$, and the third member takes problems $3, 6, 9, \cdots, 3n$.

The ICPC Seoul Regional Contest Scientific Committee must prepare a problemset consisting of $3n$ problems. The difficulty of each problem is represented by an integer from 1 to n, inclusive. For each difficulty, there are exactly three problems with that difficulty. Thus, the arrangement of difficulties in the problemset can be viewed as a *difficulty sequence* of length $3n$ containing three problems of each of the n difficulty levels.

To prevent any advantage or disadvantage for a team based on their chosen problem distribution method, the ICPC Seoul Regional Contest Scientific Committee has defined a standard called a *Fair Problemset*. A difficulty sequence of length $3n$ is called a Fair Problemset if it satisfies both of the following conditions:

1. **Sequential Distribution Fairness**: When using Sequential Distribution, for every difficulty level i $(1 \leq i \leq n)$, each of the three members receives exactly one problem with difficulty i.
2. **Jump Distribution Fairness**: When using Jump Distribution, for every difficulty level i $(1 \leq i \leq n)$, each of the three members receives exactly one problem with difficulty i.

In other words, regardless of which of the two methods is chosen, each team member must be assigned exactly one problem for each difficulty level from 1 to n, inclusive.

Given a positive integer n, write a program to find **any** Fair Problemset sequence of length $3n$.

Input

Your program is to read from standard input. The input consists of exactly one line. The line contains an integer n $(1 \leq n \leq 200$; n is **not** divisible by 3). It can be shown that for every valid input, there exists at least one Fair Problemset sequence of length $3n$.

Output

Your program is to write to standard output. Print exactly one line consisting of $3n$ space-separated positive integers, representing a Fair Problemset sequence of length $3n$. Any valid Fair Problemset sequence of length $3n$ will be accepted.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
2	1 2 1 2 1 2

Sample Input 2	Output for the Sample Input 2
4	1 2 3 4 1 4 2 3 1 3 4 2