University of Zagreb
Faculty of Electrical Engineering and Computing
Competitive Programming
Winter semester 2011 / 2012

Competitive Programming
Final Exam

4th February 2012

Dear students!
The exam (competition) consists of multiple tasks with varying difficulties. The tasks are **not** ordered by difficulty. All tasks are worth an **equal** amount of points. The level of difficulty is such that most students will not manage to solve all tasks. We hope that this will not decrease your motivation and confidence; if you are unable to solve a task, proceed to work on the other tasks.

Your program needs to read the input data from standard input (*stdin*) and output the required solution to standard output (*stdout*). Please carefully read sections INPUT and OUTPUT for each task. These sections completely specify the format of input and output data which must be **strictly adhered to** (i.e. the output must not contain additional text, such as "Solution:" etc.) for the solution to be graded as correct. This is illustrated in EXAMPLE TEST DATA sections, which contain a small number of simple input data examples with corresponding correct outputs.

Also, your programs must not access or create any files. Any violations of this rule will result in loss of points for the task.

Notice that, apart from the time limit, a memory limit is also imposed. If the program execution uses more than the allotted amount of memory (or time), it will not be graded correct.

The evaluating system is hosted on a computer running the LINUX operating system. Therefore, your program must not use DOS – specific functions. For example, the CRT module (Pascal), conio.h (C/C++) and other similar modules are forbidden.

Points are awarded solely for correct output data. The source code is not manually inspected; it is only used to derive an executable file. Only programs that output the correct solution and regularly finish their execution within time and memory limits are awarded points for the corresponding test case.

A program written in Pascal must finish execution by reaching the 'end.' statement or the 'halt' procedure, called either parameterless or as 'halt(0)'.

A program written in C/C++ must finish execution by executing 'return 0;' from the main function declared as 'int main(void)', or by calling 'exit(0)'.

For a program to score all points on a task, the underlying algorithm must be not only correct, but efficient (fast) as well. The test data is devised so that programs using correct, but less efficient, algorithms will receive partial score (for example, on a problem worth 60 points, a very slow algorithm might score 20, while a faster, but not fast enough, algorithm might score 40 points). Programs that are very fast on all test cases, but fail to produce correct outputs will, of course, score zero points. Thus, algorithm correctness is most important, while execution speed is the next priority.

Please pay attention to input data **constraints**, since they are extremely important factors when designing an appropriate efficient algorithm.

Also, please keep in mind that examples provided in EXAMPLE TEST DATA are meant to help you to understand the problem and the data formats. A program that works correctly for all the examples is **not guaranteed** to work correctly (or fast enough) on the actual test data.

Good luck!

TASK	PLODOVI	HASHMAT	USPON	IPV6	MARIO
input	standard input (*stdin*)				
output	standard output (*stdout*)				
time limit	1 second	1 second	1 second	1 second	1 second
memory limit	32 MB	32 MB	32 MB	32 MB	32 MB
points	**100**	**100**	**100**	**100**	**100**
	500 total				

PLODOVI

Author: Matija Šantl | Time limit: 1 s

Points: 100 | Memory limit: 32 MB

On a distant island, in the middle of an ocean, there lives Mister X. Since winter is coming, Mister X has been foraging for food. He has collected a number of fruits and arranged them on a drying contraption. The drying contraption has the shape of a rectangular **lattice** (grid). Each integer position in the lattice contains a single fruit, and Mister X knows the **value** of each fruit in calories. (The value can be negative in case the fruit is so spoiled that it reduces the total nutritious value of the winter stock.) A large storm is coming, and Mister X is worried that his fruits will be blown away by strong winds. That's why he is securing them with wooden sticks. The sticks can only be placed parallel to the lattice axes (horizontally or vertically) and only fruits secured by **both** a horizontal and a vertical stick are safe from the storm. Compute the maximum sum of calorie values kept after the storm in case of an optimal securing stick arrangement.

Note: Mister X can secure 0 or more fruits.

INPUT

The first line of input contains two positive integers **R** and **C** ($1 \leq \mathbf{R} \leq 10$, $1 \leq \mathbf{C} \leq 100$), the number of rows and columns, respectively, of the drying lattice.

Each of the next **R** lines contains **C** integers $\mathbf{M_{i,j}}$ ($-5000 \leq \mathbf{M_{i,j}} \leq 5000$), denoting the calorie value of the fruit in row **i** and column **j** of the drying lattice.

OUTPUT

The first and only line of output must contain the maximum possible sum of calorie values of the optimally secured fruits. (Assume that all unsecured fruits will be blown away.)

EXAMPLE TEST DATA

INPUT:
```
1 1
5
```

OUTPUT:
```
5
```

INPUT:
```
4 5
1 -1 1 -1 1
-1 2 -3 4 -5
5 -4 3 -2 1
-1 1 -1 1 -1
```

OUTPUT:
```
12
```

INPUT:
```
1 1
-5
```

OUTPUT:
```
0
```

CLARIFICATIONS

Clarification of the second test case: Mister X will secure rows 1 and 3 and columds 1, 3, and 5.

Clarification of the third test case: Mister X doesn't want to keep spoiled fruits, so it is better not to secure anything.

SCORING

In test cases worth 40% of points, **R** and **C** will be in the interval $1 \leq \mathbf{R}, \mathbf{C} \leq 8$.

HASHMAT

Author: Bruno Rahle | Time limit: 1 s

Points: 100 | Memory limit: 32 MB

You are given a hash function that is computed as follows:

$$\left(\sum_{i=1}^{n} A^{n-i+1} \cdot (\mathrm{ORD}(S_i) + B)\right) \% M$$

i.e. it is equal to the polynomial with coefficients equal to ASCII values of characters of a string (of lowercase English letters) incremented by some constant **B**, then evaluated by substituting some constant **A** as the value of **x**. The polynomial is evaluated modulo **M** (i.e. taking only the remainder of the value when divided by **M**).

For example, with **A**=2, **B**=-96 and **M**=1007, the string “banana” has the hash value of (64*2+32*1+16*14+8*1+4*14+2*1) % 1007 = 450. Given the values **A**=12, **B**=-96, **M**=101, the same string has the hash value of 57.

You are given a text and are asked to compute the hash value of **all substrings** with length exactly **K** and output them in order. For example, for the text “banana” and **K**=2, the solutions are hash values of “ba”, “an”, “na”, “an”, “na”, in that order.

INPUT

The first line of input contains 4 integers: **A** (1 ≤ **A** ≤ 1 000), **B** (-97 ≤ **B** ≤ 100), **M** (2 ≤ **M** ≤ 10 000), **K** (1 ≤ **K** ≤ 10 000; **K** ≤ |**S**|).

The second line of input contains the string **S**, consisting only of lowercase English letters and at most 100 000 characters long.

OUTPUT

If we denote the length of **S** by **L**, then the output must consist of **L**-**K**+1 lines, with line **i** containing the hash value of the substring from position **i** to **i**+**K**-1, inclusive. That is, the first line must contain the hash value of the substring from the 1st to the **K**th character, the second line from the 2nd to the (**K**+1)th, the third line from the 3rd to the (**K**+2)th, and so on until line **L**-**K**+1, containing the hash value of the substring from the (**L**-**K**+1)th to the **L**th character, inclusive.

EXAMPLE TEST DATA

INPUT:
```
2 -96 1007 2
banana
```

OUTPUT:
```
10
32
58
32
58
```

INPUT:
```
7 -93 107 3
politicar
```

OUTPUT:
```
14
38
9
84
66
51
47
```

SCORING

In test cases worth 40% of points, the product of numbers **K** and **L** will be less than 5 000 000.

USPON

Author: Dino Šantl | Time limit: 1 s

Points: 100 | Memory limit: 32 MB

It is a well-known fact that the C building has 13 floors. However, it is less well-known that elevators aren't always available. It turns into a big problem when final exams finish and all students head for the C building to view the grading results, only to finds that elevators are not running. Since such a long climb via the staircase is quite stressful, some adventurous students have decided to climb the building by scaling it from outside. You have been asked to determine the **number of different ways** to reach the **top** of the building. Since some exams are still being graded, it is **forbidden** to scale across **windows** of those rooms in order not to disturb the professors and assistants grading the exams. (Note that scaling the building from outside is not forbidden in itself, for some strange reason.)

You are given the dimensions of the building - height and width (**H** x **W**), given in windows, as well as **K** number pairs (**X**, **Y**) representing integer coordinates of windows that are **forbidden** to scale over.

At the start of the climb, the student is always at the bottom of the building, at the center by width (**W** / 2); if **W** is not even, **W** / 2 is rounded down. Thus, the starting coordinates are: **(⌊W / 2⌋, 0)**.

The student can move from the current window to the one directly above, above and to the left, or above and to the right. Mathematically, if the student is currently on the window with coordinates **(x, y)**, he can move to one of the windows given by coordinates **(x, y+1)**, **(x-1, y+1)**, **(x+1, y+1)**, of course only if the new coordinates do not denote a forbidden window or a position outside the building facade.

INPUT

The first line of input contains two integers **H** and **W** ($1 \leq$ **H**, **W** $\leq 1\,000$), the height and width of the building, separated with a single space.

The second line of input contains the integer **K** ($0 \leq$ **K** $\leq 1\,000$), the number of forbidden windows.

Each of the next **K** lines contains two space-separated integers **X** and **Y** ($0 \leq$ **X**, **Y** $\leq 1\,000$), denoting the coordinates of a single forbidden window.

OUTPUT

The first and only line of output must contain the number of different ways to reach the top of the building, **modulo** 100 007.

EXAMPLE TEST DATA

INPUT:
```
4 3
3
0 2
1 3
2 2
```

OUTPUT:
```
6
```

INPUT:
```
3 3
0
```

OUTPUT:
```
7
```

INPUT:
```
3 3
2
1 1
1 2
```

OUTPUT:
```
2
```

CLARIFICATIONS

Clarification of the first test case:

Forbidden windows are denoted by **X**, allowed windows by **O**, and the starting position of the student by **I**.

O	X	O
X	O	X
O	O	O
O	I	O

Starting coordinates: **(1, 0)**

Forbidden window coordinates: **(0, 2)** , **(1, 3)** , **(2, 2)**

Number of paths to the top: 6

Possible paths to the top:

(1, 0)-(0, 1)-(1, 2)-(0, 3),

(1, 0)-(1, 1)-(1, 2)-(0, 3),

(1, 0)-(2, 1)-(1, 2)-(0, 3),

(1, 0)-(0, 1)-(1, 2)-(2, 3),

(1, 0)-(1, 1)-(1, 2)-(2, 3),

(1, 0)-(2, 1)-(1, 2)-(2, 3).

IPV6

Author: Bruno Rahle | Time limit: 1 s

Points: 100 | Memory limit: 32 MB

Mirko the hacker lives in a future world. In that world, the Internet as we know it is completely censored. The last outposts of free and honest speech are being mown down like cannon fodder by megalomaniacal ambitions of corporations twisted by pure greed. Evil despots have forced the world to abandon the good old Internet protocol version 4 and completely switch to IPv6. Mirko has just acquired a list of IPv6 addresses owned by the aforementioned corporations. His goal is, of course, to mount an attack on these addresses.

However, the corporations own an overwhelming amount of addresses. Mirko's resources are limited, so he has decided to focus his attacks only on a subset of the addresses. He is currently busy planning the attack and recording anonymous videos involving cats, so he has asked you to determine the **contiguous interval (range) of IP addresses shared by the largest number of corporations**.

A single IPv6 address consists of **128 bits of data**, grouped as 8 segments of 4 hexadecimal digits each, separated by colon (':') signs. For example, 23a5:0201:bbdf:0000:0000:8888:0000:0000 is one such address. Since this is a long string to write, two **abbreviations** are used:

- **leading zeros in each segment can be omitted**, i.e. the IPv6 address given above can be written as 23a5:201:bbdf:0:0:8888:0:0
- **exactly one contiguous subsequence of segments that all equal 0 can be replaced by '::'**; i.e. the above address can also be written as 23a5:201:bbdf::8888:0:0 or 23a5:201:bbdf:0:0:8888::

Each IPv6 address can also be written as a single number, with the most significant segment having the largest weight (in our example, the highest weight segment is 23A5, and the IP address equals the number 47379749536320959222311704767370362880).

The list of IP addresses obtained by Mirko consists of **multiple address ranges** belonging to evil corporations. A range is given as a **pair of IP addresses** (the first and the last address in a range). For example, the range ::7f00:1 to ::7f00:100 represents all IP addresses between ::7f00:1 and ::7f00:100 (inclusive).

Mirko will attack the range of IP addresses belonging to the largest number of corporations (contained in the largest number of ranges). For example, if he was given ranges:

- ::7f00:1 to ::7f00:100
- ::1 to 1::
- ::7f00:50 to ::7fff:0

Mirko would attack the range ::7f00:50 to ::7f00:100 since it is a subrange of 3 ranges.

INPUT

The first line of input contains the positive integer **N** ($1 \leq$ **N** $\leq 100\ 000$), the number of ranges on the list.

Each of the next **N** lines contains a single **pair of IPv6 addresses** representing a single range (a range belongs to one company). All IPv6 addresses will be valid, and all hexadecimal digits will be lowercase. The first address in a range will always be smaller than the second.

Warning: The test inputs will be relatively large. Make sure that you are not using slow input methods.

OUTPUT

The first and only line of output must contain the range of IPv6 addresses described above. If there are multiple solutions, output the one with the smallest starting IP address. **Output all IPv6 addresses in full form (without abbreviations).**

EXAMPLE TEST DATA

INPUT:

```
3
::7f00:1 ::7f00:100
::1 1::
::7f00:50 ::7fff:0
```

OUTPUT:

```
0000:0000:0000:0000:0000:0000:7f00:0050 0000:0000:0000:0000:0000:0000:7f00:0100
```

INPUT:

```
5
0000:0000:0000:0000:0000:0000:0000:0001 0000:0000:0000:0000:0000:0000:0000:0011
0000:0000:0000:0000:0000:0000:0000:0003 0000:0000:0000:0000:0000:0000:0000:0004
0000:0000:0000:0000:0000:0000:0000:0000 0000:0000:0000:0000:0000:0000:0000:000e
0000:0000:0000:0000:0000:0000:0000:0009 0000:0000:0000:0000:0000:0000:0000:000d
0000:0000:0000:0000:0000:0000:0000:0001 0000:0000:0000:0000:0000:0000:0000:0005
```

OUTPUT:

```
0000:0000:0000:0000:0000:0000:0000:0003 0000:0000:0000:0000:0000:0000:0000:0004
```

INPUT:

```
5
5cc.3140::3af 1c83::16f
e8f5.f7ff.62db.ff6b.ab80:: f000::e
3545.3203.5000::13bc 8c7c.32ae.f1e.4fa0::3c.1b6d.1736
4b63.e630:: b000::eb.59f9
a489.9910:: ac9c.4000::d8c
```

OUTPUT:

```
4b63.e630.0000:0000:0000:0000:0000:0000 8c7c:32ae:0f1e:4fa0:0000:003c:1b6d:1736
```

SCORING

In test cases worth 40% of points, all IP addresses will use (have nonzero) only the last 32 bits of data.

In test cases worth 60% of points, all IP addresses will use (have nonzero) only the last 64 bits of data.

In test cases worth 50% of points, all IP addresses will be given in full form (without using abbreviations).

In test cases worth 40% of points, the number **N** will be less than 1000.

MARIO

Author: Ivan Katanić | Time limit: 1 s

Points: 100 | Memory limit: 32 MB

Mirko has coded yet another spin-off of the Super Mario game. In his version there are **N** columns of different heights arranged in a row, denoted by numbers from **1** to **N** from left to right. Mario's starting position is the leftmost column. His goal is reaching the rightmost column, where Princess Peach is imprisoned. In a single move, Mario can jump on any column to the right of the current one that is **strictly higher** than the current column. The first (leftmost) column is always the **shortest**, and the goal (rightmost) the **tallest** one.

Mirko would like to know the **number of different ways** to finish the game, so he has naturally asked you for help. Since the number can be extremely large, Mirko is only interested in the remainder modulo 10 007.

INPUT

The first line of input contains the positive integer **N** ($1 \leq \mathbf{N} \leq 100\,000$).
The next **N** lines of input contain column heights, one per line, from left to right. The heights are positive integer less than 1 000 000 (one million).

OUTPUT

The first and only line of output must contain the number of ways that Mario can reach the last column (and save the Princess), modulo 10 007.

EXAMPLE TEST DATA

INPUT:
```
2
5
9
```

OUTPUT:
```
1
```

INPUT:
```
3
1
2
3
```

OUTPUT:
```
2
```

INPUT:
```
6
4
9
6
9
7
10
```

OUTPUT:
```
7
```

CLARIFICATIONS

Clarification of the first test case: The only way to the Princess is jumping from the first to the second column.

Clarificaton of the second test case: One way to the Princess is jumping directly from the first to the third column, and the other way is via the second column.

SCORING

In test cases worth 50% of points, the number **N** will be less than or equal to 1000.