DAY 1
Overview

Author	Carl	CEOI	Bruce
Problem	**Budget**	**Hanoi**	**Domino**
Program name	`budget`	`hanoi`	`domino`

Source name	`budget.pas`
	`budget.java`
	`budget.cpp`
		`hanoi.pas`
		`hanoi.java`
		`hanoi.cpp`
			`domino.pas`
			`domino.java`
			`domino.cpp`

Input file	`budget.in`	`hanoi.in`	`domino.in`
Output files(10)	`budget.out`	`hanoi.out`	`domino.out`
Time limit	10 second	1 second	1 second
Num. of tests	10	10	10
Points per test	10	10	10
Total points	**100**	**100**	**100**

The maximum total score for Day1 is 300 points.

Budgeting Programmers

Author
Carl Hultquist

Introduction

Bruce has risen up the corporate ladder in the company for which he works, and has been put in charge of a bunch of programmers. There are a lot of jobs that need to be done by the programmers, and it's Bruce's job to assign jobs to the various programmers.

However, there is a small problem. Because the programmers are skilled in different areas, they will take different amounts of time to complete any given job. They might also make mistakes: this will often happen if a programmer is given a job that deals with a technology that they aren't familiar with or that bores them.

Before assigning each programmer their jobs, Bruce sends out a list of the jobs to everyone and asks them to tell him how much time they think it would take them to do each job. As the manager, Bruce knows roughly how many mistakes each programmer might make on a particular job, and he makes a table of this information. By putting the information from his programmers together with the table he has made, Bruce creates a table that shows:

- How long a programmer will take to do a job
- How many mistakes a programmer will make in doing a job

Of course, the easiest way of solving this would be to give jobs to the programmers that make the fewest mistakes. But Bruce only has a certain amount of money to pay his programmers, meaning that he can only pay for a certain maximum number of hours of total work.

Task

Given the maximum number of hours that the programmers can work in total, you must find the fewest number of mistakes that can be made by assigning programmers to jobs.

Example

In this example Bruce has 3 programmers, 4 jobs to be done, and he can pay for a total of 7 hours of work. Here is the table that Bruce has put together:

	Programmer 1	**Programmer 2**	**Programmer 3**
Job 1	1 hour, 4 mistakes	2 hours, 3 mistakes	3 hours, 0 mistakes
Job 2	3 hours, 2 mistakes	2 hours, 5 mistakes	1 hour, 7 mistakes
Job 3	2 hours, 7 mistakes	1 hour, 1 mistake	3 hours, 3 mistakes
Job 4	2 hours, 4 mistakes	2 hours, 2 mistakes	2 hours, 1 mistake

In this case, the best way of assigning the jobs to the programmers would be to give jobs 1 and 2 to programmer 1, job 3 to programmer 2 and job 4 to programmer 3. This gives a total of 7 hours of work, and the programmers make 8 mistakes in total.

Input (budget.in)

The first line of **budget.in** will contain three integers: **N**, **M** and **H**. **N** is the number of programmers, **M** is the number of jobs that need to be done, and **H** is the total number of hours of work that can be paid for. The next **M** lines of input will contain **N** pairs of integers. The $\mathbf{i}^{th}$ pair of integers on the $\mathbf{j}^{th}$ line of input indicates how well programmer **i** can do job **j**. The first integer in the pair indicates how long the programmer will take to do the job (in hours), and the second indicates how many mistakes the programmer will make in doing that job.

Sample Input:

```
3 4 7
1 4 2 3 3 0
3 2 2 5 1 7
2 7 1 1 3 3
2 4 2 2 2 1
```

Output (budget.out)

The first and only line of **budget.out** must contain a single integer, **T**, which is the smallest total number of mistakes that will be made by the programmers.

Sample output:

```
8
```

Constraints

- 2 • **N** • 100
- 2 • **M** • 1000
- M • **H** • 5000
- 1 • **t** • H, where **t** is the amount of time it takes a programmer to perform a job
- 1 • **T** • 1000000
- No programmer makes more than 100000 mistakes in any job
- Time limit 6 seconds
- A solution is always possible.

Towers of Hanoi

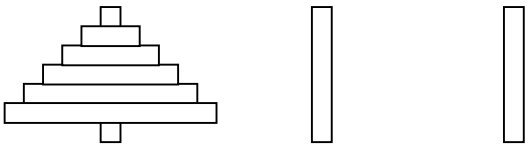
Author

Central European Olympiad in Informatics 2003

Introduction

You may be familiar with the legend of the towers of Hanoi. A group of monks were given three pegs, with 64 discs placed on one peg. They must move all the discs to another peg by moving them one at a time. However, the discs are all different sizes, and they may only move a disc onto an empty peg or onto a larger disc (initially the disc are stacked with the largest at the bottom). When they have completed the task, the universe will end.

Unfortunately, the monks lost track of their strategy part way through, and for a while have been making moves somewhat at random. They need your help to get back on track.



Task

You will be given the current position of the *N* discs on the three pegs (the problem considers the general case of *N* discs, not just 64). Compute the minimum number of moves required to move all the discs onto peg 3. This number may be very large (after all, the monks will spent the rest of time completing the movements), so you need only calculate the last 6 digits (i.e. the remainder after division by 1 000 000).

Example

There are 7 discs, numbered 1 to 7 in increasing order of size. Peg 1 contains discs 1 and 2, peg 2 contains disc 3 and peg 3 contains the remaining discs. Then the best strategy is to move disc 3 to peg 3, disc 1 to peg 2, disc 2 to peg 3 and finally disc 1 to peg 3.

Input (hanoi.in)

The first line of input contains *N*, the number of discs. The second line contains three space-separated integers, which are the number of discs on pegs 1, 2 and 3. The third line contains the numbers of the discs on peg 1 as a list of space-separated integers, in decreasing order of size. The fourth and fifth lines similarly describe pegs 2 and 3. Note that a peg may have no discs, in which case a blank line appears in the input.

Sample Input:

```
7
2 1 4
2 1
3
7 6 5 4
```

Output (hanoi.out)

The output consists only of a single integer. This is the remainder when the minimum number of moves required is divided by 1 000 000.

Sample output:

```
4
```

Constraints

1 • *N* • 100 000

Time limit

1 second.

Scoring

You will score 100% for a correct answer, and 0% for an incorrect answer.

Domino

Author

Bruce Merry

Introduction

Dr. Evil has imprisoned you in his underground lair. He has learnt from his previous mistakes with Austin Powers, and this time will not place you in an easily escapable situation and not even watch. He will place you in a more difficult to escape situation, but will still not watch. You can only escape the room you are in by solving a puzzle involving dominoes.

Task

You are given a set of dominoes. Each domino contains two square halves, with 0 to $N - 1$ dots (inclusive). The set contains one of each possible domino, for a total of $N(N + 1) / 2$ (this includes dominoes with the same number of dots on each half). On the floor of the room is a rectangle which is just big enough to contain all the dominoes, divided into squares such that each domino covers two squares. Each square contains indentations, which must match up to the dots on one half of a domino (assume that the dots will match up as long as there are the right number of them). If you match up all the dots on the dominoes to the squares in the rectangle, then the door will open and you can escape.

Given N and the patterns of indentations, figure out how to place all the dominoes so that you can escape. It is guaranteed that it is possible to escape.

Example

Suppose N is 3. Then the dominoes are (0,0), (0,1), (0,2), (1,1), (1,2) and (2,2). Suppose the indentations on the floor are as follows:

1	1	0	1
0	1	2	0
0	2	2	2

Then the dominoes can be placed like this:

1	1	0	1
0	1	2	0
0	2	2	2

Input (domino.in)

The first line contains a single integer N. The second line contains two space-separated integers, R and C. The rectangle is R by C squares, and $RC = N(N + 1)$. The remaining R lines each describe one row of the rectangle, from top to bottom. Each line contains C space-separated integers, each representing the number of indentations within a square, from left to right.

Sample Input:

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

Output (domino.out)

The output contains R lines of C space-separated integers, corresponding to the description of the rectangle in the input file. The dominoes must be numbered from 1 to $N(N + 1) / 2$ (in any order), and the squares in the output file must be labelled with the number of the domino covering them. Thus each domino is represented by a pair of adjacent and equal numbers in the output file.

Sample output:

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

Constraints

$1 \bullet N \bullet 13$

Time limit

1 second.

Scoring

- You will score 0% for an incorrect answer and 100% for a correct answer.