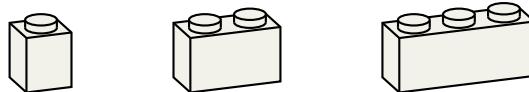


## 1 Problem

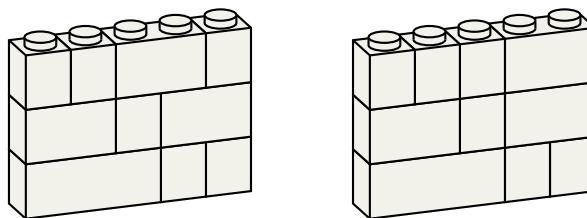
You have an infinite supply of three different sizes of toy bricks: (a)  $1 \times 1 \times 1$ , (b)  $2 \times 1 \times 1$  and (c)  $3 \times 1 \times 1$  (width, height and depth). They are shown below:



Toy bricks are always used in upright position. The faces of the bricks are made from identical material and are indistinguishable. As famously known, the bricks have hollows tubes on their bottom and studs on their top. Two bricks can be *locked* through these tubes/studs, by placing one of them on top of the other (such that at least one tube/stud pair is coincident).

We want to construct a thin upright wall of width  $W$  and height  $H$  using these bricks. Such a wall always has depth 1. The wall is fully filled with bricks; it contains no holes in its interior or on its boundary.

We also want the wall to be *connected*: All bricks are directly or indirectly (through a path of directly locked bricks) locked to each other. To exemplify, we show two walls of size  $5 \times 3$  below: one connected (on the left) and one not connected (on the right).



In how many different ways can we construct such a connected wall? Write a program that answers this question given  $W$  and  $H$ . Since the number of ways can be quite large, your program is supposed to answer this question modulo 8191.

Note that the front face of the wall carries significance while counting. A wall and its mirror image (equivalently, its  $180^\circ$  rotation) are distinct configurations, unless the wall structure is symmetric along its width.

## 2 Input Format

W H

W = Width of the wall. H = Height of the wall.

## 3 Output Format

N

N = The number of ways to construct such a wall modulo 8191.

## 4 Limits

$$1 \leq W, H \leq 100\,000$$

For any input, either  $W^2$  or  $W \times H^4$  is less than 100 000 000.

Time Limit: 1 second  
Memory Limit: 256 MB

## 5 Clarifications

- Your solution is expected as a C++ program source named `1the2.cpp` that reads from the standard input and writes to the standard output.
- It is OK to copy code from the sample codes we shared in our course website in ODTÜClass. You can also copy code from your previous THE submissions for this course. Copying from elsewhere will be considered cheating.
- You are supposed to submit your code via the VPL item in ODTÜClass. Use the “evaluate” feature to run the auto-grader on your submission.
- The grade from the auto-grader is not final. We can later do further evaluations of your code and adjust your grade. Solutions that do not attempt a “reasonable solution” to the given task may lose points.
- We will compile your code with `g++` using the options: `-std=c++2a -O3 -lm -Wall -Wextra -Wpedantic`
- Late submissions are not allowed.

## 6 Sample Input/Output Pairs

Input	Output
2 8	255
3 4	225
6 4	1860
8 2	148
11 11	5202

Input	Output
23 32	6167
100 100	2097
1000 1000	3616
1000 100000	4498
100000 4	1769

## 7 Hints

- A well-implemented combination of the two algorithms that we described in the class is sufficient to get full points. (Choose the right algorithm at the start.)
- Slower approaches (such as an exhaustive search) may get partial points.
- Recall the rules of modular arithmetic (for fixed modulo divisor). If a mathematical expression is made up of addition, subtraction, and multiplication, then the modulo operation can be (redundantly) applied to any of its subexpressions, without affecting the expression’s overall modular value.
- Please, pay attention to the bounds of your data types. If you get an overflow/underflow before you can apply the modulo operation (%), you will be in trouble. Analyze the potential bounds of your expressions and apply modulo as necessary to avoid overflows/underflows. We also suggest trying to be precise while being safe: avoid unnecessary modulo operations for better efficiency.
- The modulo in C++ is not a mathematical modulo. It works differently for negative numbers. Ideally, your expressions should not produce negative values before applying the modulo. If such expressions are unavoidable, you can make them non-negative by adding the modulo divisor to their value.