Task 1. Monopoly

Deni enjoys playing the game monopoly very much. But the duration of one game is very long – from 5 to 7 hours. So, **D**eni starts thinking about changing the classical rules. The players in monopoly usually move in one direction from space to space and after some time they return to the beginning, start again and so on. In the new version, the movement will again be from space to space but from the current one there can be several possibilities for the next move. **D**eni wants to find such directed connections between the spaces, so that a player can never return to a space, where he has been, no matter how he moves (of course, following the rules). In such a way the game will be shorter in duration.

She has already begun making the new board – she has chosen the number of spaces ***N*** (the spaces are numbered from 1 to ***N***) and she has made a list with ***M*** connections (each connection has a direction and there is no connection that connects a space with itself). If space *i* is connected to space *j*, then there is no direct connection in the opposite direction, i.e. from space *j* to space *i*, and also, there are no other direct connections from space *i* to space *j*. **D**eni thought that she was ready with the new board, but suddenly she noticed that the condition she wants (when you move from space to space, using the connections, you cannot return to a previously visited space) doesn't hold for her list of connections. She first thought to remove some of the direct connections, but that will result in rewriting the list, which can be really long. That's why **D**eni decided to reverse the direction of some of the directed connections.

Task

You are regularly playing monopoly with **D**eni. That's why you want to help her by writing the program **monopoly**, which has to tell her which connections to reverse so that the stated condition will hold. The program has to contain the function *find_reverse* which will be compiled with the jury's program.

Implementation details

The function *find_reverse* must have the following prototype:

std::string find_reverse (int N, int M, int connections[][2]);

It is called only once by the jury's program with three parameters: ***N*** – the number of spaces in the new board, ***M*** – the number of directed connections in **D**eni's list and the array *connections* which has ***M*** rows, each consisting of two numbers *x* and *y* – the beginning and ending spaces for the directed connection. This function should return a binary string of length ***M*** – in the order of the array *connections*, for each connection you have to put '1' in the string, if the connection has to be reversed and '0', otherwise. If there is more than one solution, you can return any of them.

You have to submit to the system the file **monopoly.cpp**, which contains implementation of the function *find_reverse*. The file may contain other functions and code, needed for your program, but it **must not contain** a main function – *main*. Also, you should not try to read from the standard input nor try to write to the standard output!

Constraints

- $3 \le N \le 5 \times 10^5$
- $3 \le M \le 1.5 \times 10^6$

Subtasks

Subtask	Points	*N*	*M*	Further constraints
1	0	–	–	The example
2	15	≤ 7	≤ 21	–
3	40	$\leq 10^3$	$\leq 5 \times 10^3$	–
4	25	$\leq 10^5$	$\leq 5 \times 10^5$	–
5	20	$\leq 5 \times 10^5$	$\leq 1.5 \times 10^6$	–

In order to get the points for a given subtask, your solution must pass all the tests for the subtask.

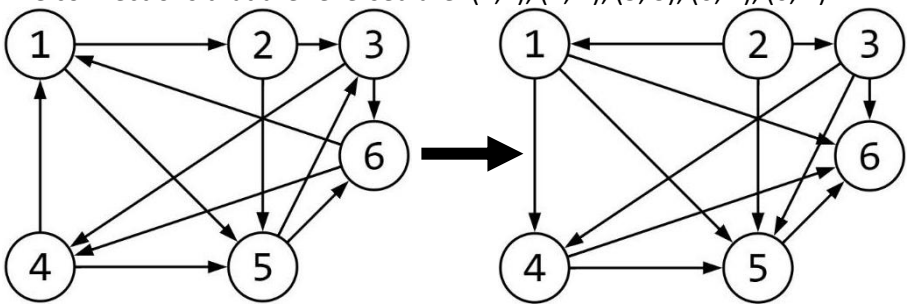
Local testing

For local testing you are given the file **Lgrader.cpp**. You have to place it in the same folder as your program **monopoly.cpp** and you have to compile together the files **Lgrader.cpp** and **monopoly.cpp**. In such a way, you will make a program to check the correctness of your function. The program will require the following sequence of data from the standard input:

- on the first line: two integers – the number of spaces ***N*** and the number of connections ***M*** of the new board.
- on the last ***M*** lines: on each line there have to be two integers *x* and *y* – the beginning and ending spaces for each directed connection.

The output of the program will be the binary string that you found.

Example of local testing

Input	Output	Explanation
6 12 1 2 1 5 2 3 2 5 3 4 3 6 4 1 4 5 5 3 5 6 6 1 6 4	100000101011	The connections that are reversed are: (1,2), (4, 1), (5, 3), (6, 1), (6, 4).  The figure above is showing the spaces, firstly, with the directed connections from **D**eni's list and, secondly, after reversing the direction of the connections with corresponding '1' in the output. Clearly, in the beginning using the directed connections (1, 2), (2, 3), (3, 4) and (4, 1) we can start from space 1 and moving along these directed connections, we will return to that space. One can see that **D**eni's condition holds in the resulting board. Notice that there are other valid solutions: (4, 1), (5, 3), (6, 1), (6, 4) (1, 5), (2, 3), (2, 5), (3, 4), (3, 6), (4, 5), (5, 6)