Missing Number

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

You are given an array of n integers from 1 to $n+1$. It is possible that some of the integers appear multiple times in the array. Still, it is clear that at least one of the integers from 1 to $n+1$ is missing from the array. Your task is to find *any* such integer. Normally, it would be easy, but there is an unfortunate issue: there is not enough memory on your computer to represent the whole array.

Hence, you are asked to produce a *streaming* algorithm instead. Informally, the elements of the array are given to you one-by-one with no way to randomly access them. However, there is one solace: after receiving the last element of the array, you can restart reading it from the beginning. Ideally, you would like to avoid such a situation and solve the whole problem in one pass. However, sometimes (including this very problem) multiple passes over the input may be necessary to avoid using too much memory. You need to balance both concerns and *simultaneously* use not too much memory and make not too many passes over the array (the problem is trivial if you are allowed to store n bits of memory or use n passes).

There is still one catch, however. For technical reasons, you need to produce a *deterministic finite automaton* (DFA) over the alphabet $\{0, 1, \ldots, n+1\}$ implementing the logic of your algorithm. The characters of the alphabet correspond to the information you receive when reading the stream: numbers from 1 to $n+1$ represent reading a corresponding integer from the array, while the number 0 represents the fact that you have reached the end of the array. After the first 7 times you have reached the end of the array, the array will be given to you again in the same order as before, always ending with a 0 character representing the end of the array. Hence, your DFA will be fed the string $a_1, a_2, \ldots, a_n, 0$ a total of 8 times (here, $a_1, a_2, \ldots, a_n$ is the hidden array). Finally, at any moment of the execution, your automaton can "say" that it already knows some correct answer to the problem; in that case, the execution is halted. If your DFA has not done so by the time it has read the whole stream (including the 0 character at the end) 8 times, it is not accepted as a solution. Similarly, if it halts, but with an incorrect answer, it is not accepted as a solution either.

Let us state the problem formally. There is a hidden array $a_1, a_2, \ldots, a_n$ of n integers from 1 to $n+1$. In the example, $n = 4$, in all other tests $n = 250$. You need to specify a finite state machine with at most 4000 states, numbered from 0 to $q-1$, where $q \le 4000$ is the number of states. For the state i ($0 \le i \le q-1$), you need to specify $n+2$ numbers $\delta(i, 0)$, $\delta(i, 1)$, ..., $\delta(i, n+1)$. The number $\delta(i, j)$ represents the behavior of your DFA when it is in the state i and reads the number j from the input. It must be an integer from $-(n+1)$ to $q-1$ inclusive. Negative integers correspond to the fact that the execution is halted and an answer is given; specifically, $-s$ corresponds to your DFA claiming that $+s$ is a correct answer to the problem. Nonnegative integers correspond to the fact that the execution is continued and the state of the automaton changes from i to $\delta(i, j)$. The execution of the automaton starts in the state 0.

Your DFA is considered to work correctly on the array $a_1, a_2, \ldots, a_n$ if it halts with a correct answer at some point of reading the string $a_1, a_2, \ldots, a_n, 0, a_1, a_2, \ldots, a_n, 0, \ldots, a_1, a_2, \ldots, a_n, 0$, where the array is repeated 8 times. Your DFA is considered to work correctly in general if it works correctly on all possible arrays $a_1, a_2, \ldots, a_n$ of integers from 1 to $n+1$ (hence, probabilistic solutions are discouraged).

For $n = 4$, we will check your DFA on all possible arrays. For $n = 250$, we do not have means to check it on all arrays, so we will check it on less than 5000 specially selected ones. There are two tests in the problem; both tests effectively check that your approach works correctly on many arrays simultaneously.

Input

One integer n, which is either equal to 4 or to 250.

Output

On the first line, print a single integer q ($1 \leq q \leq 4000$) denoting the number of states in your DFA.

On the i-th of the following q lines, print $n+2$ integers $\delta(i-1,0)$, $\delta(i-1,1)$, ..., $\delta(i-1,n+1)$. They should all be from $-(n+1)$ to $q-1$ inclusive and correspond to the transitions of the automaton. Note that **all** numbers should be in this range: even if some combination of the state i and the input number j can never be reached on any valid input, the value of $\delta(i,j)$ that you specify should still be in the range $[-(n+1), q-1]$.

Example

standard input	*standard output*
4	8 -3 0 0 1 0 0 2 1 1 1 1 1 -1 3 2 2 2 2 4 3 3 3 3 3 -5 4 4 4 4 5 6 5 5 5 5 5 -2 6 7 6 6 6 -4 7 7 7 7 7

Note

The answer for the example is a very boring automaton that uses four passes to check whether the array includes numbers 3, 1, 5, and 2 (in that order). Each of the passes effectively uses only two states that correspond to whether it has already found the number it is looking for in the current pass. If it reaches the end of the array (the character 0) without finding the necessary number, it immediately halts, saying that the necessary number is indeed missing. Otherwise, it proceeds to use the next pass to search for the next number. Finally, if it has found all four necessary integers, then the number 4 has to be missing. Hence, it does not do a fifth pass looking for the number 4 (but it could have without violating the restrictions on the numbers of the automaton states and used passes).

Of course, this naive approach has no chance of working for $n = 250$, and something much better is needed. The main reason for the example's existence is to illustrate the output format.