Problem 5. Slots

Input file:	`input.bin`
Output file:	`output.bin`
Time limit:	1 second (1.5 seconds for Java)
Memory limit:	256 mebibytes

Game programming has its own tips and tricks for solving commonly occuring problems; these tricks are called "patterns". One of the patterns has to do with placing game objects within the array.

There is an array A of fixed size N consisting of *slots*, numbered from 1 to N. Throughput the game, game *objects* can be created and destroyed. From the moment of its creation, each object must be assigned into a slot in the array A. When an object is destroyed, its slot becomes free and can be eventually reused for newly created objects.

Moreover, each object also has an integer *ID*, which remains unique for the object throughout the game. IDs are assigned to objects consecutively starting from the number 1.

Finally, to assure that the creation of new objects does not take longer than $O(1)$, a stack of all currently free slots of the array A is maintained.

A new object is created in several steps:

1. The slot index x is popped from the top of the free slots stack.
2. The new object is assigned to the slot x.
3. The new object is assigned an ID, which is either greater by 1 than the last assigned ID, or equals 1 if it is the very first object in the game.

An object is destroyed in the following steps:

1. The slot x, which contains the object being destroyed, becomes free.
2. The slot index x is pushed onto the top of the free slots stack.

Such a system allows to emplace game objects into the array of slots conveniently. The operations of creation and destruction both take constant time. Moreover, we can even store "weak references" to objects thanks to having IDs! Weak reference as such a reference that you can check whether the object it points to has already been destroyed or is still alive.

In this problem, you must find out whether it is possible to reach the given configuration of the described system of slots; and if so, how to do that with the least possible number of operations. Two types of operations are allowed: create a new object and destroy the object in the given slot.

For each slot it is known whether it must be free or occupied in the end of the game. If a slot must be occupied, then an object with the prescribed ID must be located in it.

Initially all slots are free, as there are no live objects and the game has just begun. The free slots stack contains all the slots of array A in their natural order; the first slot is at the top of the stack.

Attention: in this problem, the input file **input.bin** and the output file **output.bin** contain **binary data**! All integers are provided in the format native to the checking machines, with little-endian byte order.

Input

The input file consists of $(N + 1)$ 32-bit integers. The first integer is N — the number of slots ($1 \le N \le 10^6$). The remaining N numbers define the required configuration.

Each number defines the contents of the corresponding slot:

- 0 — means that the slot must be free,
- $k > 0$ — means that the slot must contain an object with ID = k.

It is guaranteed that all given IDs are different and do not exceed $2 \cdot 10^6$.

Output

If the required configuration cannot be obtained, the output file must contain a single 32-bit integer -1.

Otherwise print $(M + 1)$ 32-bit integers. The first of these integers is M — the minimal number of operations necessary to produce the configuration. The remaining M numbers define the operations in the order of their execution.

Each operation is encoded as a single integer:

- 0 — means that a new object must be created,
- $1 \le x \le N$ — means that the object in the slot x must be destroyed.

All operations must be correct: you cannot destroy an object in a free slot; you cannot create an object when there are no free slots.

Examples

These examples show the contents of the input and output files in hex format. In the testing system, files will contain binary data. Examples in binary format can be downloaded in the ¡¡News¿¿ tab near the problem statements.

input.bin
0A 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 0A 00 00 00 09 00 00 00 08 00 00 00 00 00 00 00 0C 00 00 00
output.bin
0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 09 00 00 00

input.bin
05 00 00 00 05 00 00 00 04 00 00 00 03 00 00 00 02 00 00 00 01 00 00 00
output.bin
FF FF FF FF

Example explanation

In the first example, we consecutively create 8 objects, which occupy the first 8 slots, with the IDs from 1 to 8. Next, the objects in the slots 6 and 7 are destroyed (with IDs of 6 and 7, respectively), in such order that the slot 7 ends up at the top of the free slots stack. Next, two objects are created: they are assigned IDs 9 and 10, which are next in order, and fall into the slots 7 and 6, respectively. Finally, two more objects with IDs 11 and 12 are created, which are assigned to the last two slots, and the object with ID 11 in the slot 9 is destroyed.

In the second example, it is impossible to achieve the required configuration.