UVa Online Judge

11563 Introspective Caching

In a distributed system, data is never where you need it, and fetching data over a network takes time and consumes bandwidth. The problem can be mitigated by adding a cache, where a node stores some resources locally and if those resources need to be used again, it can simply take them from its cache rather than asking someone else for them.

However, caches have a nasty tendency to fill up, so at some point, objects must be evicted from the cache to make room for new objects. Choosing what object to remove from the cache is not easy and there are several different algorithms to choose from.

The marvelous Apes in Computing Machinery have come up with a fantastic new algorithm, the Introspective Caching Algorithm, named after a city of Peru. It consists of some extra hardware (a very small, precognitive monkey) which helps making decisions. Since the monkey can see into the future, she knows exactly what objects will be accessed and in what order, and using this information she will make optimal decisions on what objects to remove from the cache. Optimality here means that she will minimize the number of times an object is read into the cache.

All object accesses go through the cache, so every time an object is accessed, it must be inserted into the cache if it was not already there. All objects are of equal size, and no writes occur in the system, so a cached object is always valid. When the system starts, the cache is empty.

You have been tasked with evaluating the monkey's performance, and feeding her the occasional banana.

Input

The input file contains several test cases, each of them as described below.

The first line of input contains three integers, separated by single spaces, telling you how many objects fit in the cache, $0 < c \leq 10000$, how many different objects are in the system, $c \leq n \leq 100000$, and how many accesses, $0 \leq a \leq 100000$, will occur. The following a lines contain a single integer between 0 and $n - 1$ (inclusive) indicating what object is accessed. The first line corresponds to the first object accessed access and the last line to the last.

Output

For each test case, write to the output the least number of times an object must be read into the cache to handle the accesses listed in the input, on a line by itself.

Sample Input

```
1 2 3
0
0
1
3 4 8
0
1
2
3
```

3
2
1
0

Sample Output

2
5