C3. Smart Beaver and Resolving Collisions

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The Smart Beaver from ABBYY has a lot of hobbies. One of them is constructing efficient hash tables. One of the most serious problems in hash tables is resolving collisions. The Beaver is interested in this problem very much and he decided to explore it in detail.

We assume that the hash table consists of h cells numbered from 0 to h - 1. Objects are added to and removed from it. Every object has its own unique identifier. In addition, every object has a corresponding hash value — an integer between 0 and h - 1, inclusive. When an object is added to the table, if the cell corresponding to the hash value of the object is free, then this object goes there. If the cell is already occupied by another object, there is a collision. When an object is deleted from the table, the cell which it occupied becomes empty.

The Smart Beaver has recently learned about the method of linear probing to resolve collisions. It is as follows. Let's say that the hash value for the added object equals t and cell t of the table is already occupied. Then we try to add this object to cell $(t + m)\ mod\ h$. If it is also occupied, then we try cell $(t + 2 \cdot m)\ mod\ h$, then cell $(t + 3 \cdot m)\ mod\ h$, and so on. Note that in some cases it's possible that the new object can not be added to the table. It is guaranteed that the input for this problem doesn't contain such situations.

The operation $a\ mod\ b$ means that we take the remainder of the division of number a by number b.

This technique immediately seemed very inoptimal to the Beaver, and he decided to assess its inefficiency. So, you are given a sequence of operations, each of which is either an addition of an object to the table or a deletion of an object from the table. When adding a new object, a sequence of calls to the table is performed. Calls to occupied cells are called dummy. In other words, if the result of the algorithm described above is the object being added to cell $(t + i \cdot m)\ mod\ h\ (i \geq 0)$, then exactly i dummy calls have been performed.

Your task is to calculate the total number of dummy calls to the table for the given sequence of additions and deletions. When an object is deleted from the table, assume that no dummy calls are performed. The table is empty before performing the operations, that is, initially it doesn't contain any objects.

Input

The first line of input contains three integers h, m and n $(1 \leq m < h)$, separated by spaces, where h is the size of the hash table, m is the number that is used to resolve collisions, n is the number of operations.

The following n lines contains the descriptions of the operations. Their execution order corresponds to the order in which they appear in the input file. Each operation is described by a single line. The operations are described as follows:

- "`+ id hash`"
  This is the format of the operation that adds an object to the table. The first character is "+" (ASCII 43), followed by a single space, then the object identifier id $(0 \leq id \leq 10^9)$, then another space, and the hash value of the given object $hash$ $(0 \leq hash < h)$. The object identifier and the hash value of this object are integers.
- "`- id`"
  This is the format of the operation that deletes an object from the table. The first character is "−" (ASCII 45), followed by a single space, then the object identifier id $(0 \leq id \leq 10^9)$. The object identifier is an integer.

It is guaranteed that for all addition operations the value of id is unique. It is also guaranteed that the initial data is correct, that is, it's always possible to add an object to the hash table and there won't be any deletions of nonexisting objects.

The input limitations for getting 20 points are:

- $1 \le h \le 5000$
- $1 \le n \le 5000$

The input limitations for getting 50 points are:

- $1 \le h \le 5 \cdot 10^4$
- $1 \le n \le 5 \cdot 10^4$

The input limitations for getting 100 points are:

- $1 \le h \le 2 \cdot 10^5$
- $1 \le n \le 2 \cdot 10^5$

Output

Print a single number — the total number of dummy calls to the hash table.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use `cin`, `cout` streams and the `%I64d` specifier.

Examples

input
10 2 7 + 11 0 + 22 2 + 33 6 + 44 0 + 55 0 - 22 + 66 0

output
7

input
5 1 6 + 123 0 + 234 1 + 345 2 - 234 + 456 0 + 567 0

output
4