

E. Martian Luck

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You know that the Martians use a number system with base k . Digit b ($0 \leq b < k$) is considered *lucky*, as the first contact between the Martians and the Earthlings occurred in year b (by Martian chronology).

A *digital root* $d(x)$ of number x is a number that consists of a single digit, resulting after cascading summing of all digits of number x . Word "cascading" means that if the first summing gives us a number that consists of several digits, then we sum up all digits again, and again, until we get a one digit number.

For example, $d(3504_7) = d((3 + 5 + 0 + 4)_7) = d(15_7) = d((1 + 5)_7) = d(6_7) = 6_7$. In this sample the calculations are performed in the 7-base notation.

If a number's digital root equals b , the Martians also call this number lucky.

You have string s , which consists of n digits in the k -base notation system. Your task is to find, how many distinct substrings of the given string are lucky numbers. Leading zeroes are permitted in the numbers.

Note that substring $s[i...j]$ of the string $s = a_1 a_2 \dots a_n$ ($1 \leq i \leq j \leq n$) is the string $a_i a_{i+1} \dots a_j$. Two substrings $s[i_1...j_1]$ and $s[i_2...j_2]$ of the string s are different if either $i_1 \neq i_2$ or $j_1 \neq j_2$.

Input

The first line contains three integers k , b and n ($2 \leq k \leq 10^9$, $0 \leq b < k$, $1 \leq n \leq 10^5$).

The second line contains string s as a sequence of n integers, representing digits in the k -base notation: the i -th integer equals a_i ($0 \leq a_i < k$) — the i -th digit of string s . The numbers in the lines are space-separated.

Output

Print a single integer — the number of substrings that are lucky numbers.

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

Examples

input	Copy
10 5 6 3 2 0 5 6 1	
output	Copy
5	

input	Copy
7 6 4 3 5 0 4	
output	Copy
1	

input	Copy
257 0 3 0 0 256	
output	Copy
3	

Note

In the first sample the following substrings have the sought digital root: $s[1 \dots 2] = "3\ 2"$, $s[1 \dots 3] = "3\ 2\ 0"$, $s[3 \dots 4] = "0\ 5"$, $s[4 \dots 4] = "5"$ and $s[2 \dots 6] = "2\ 0\ 5\ 6\ 1"$.