Dr. Tuple is working on the new data-mining application for Advanced Commercial Merchandise Inc. One of the subroutines for this application works with two arrays P and Q containing N records of data each (records are numbered from 0 to $N-1$). Array P contains hash-like structure with keys. Array P is used to locate record for processing and the data for the corresponding record is later retrieved from the array Q.

All records in array P have a size of S_P bytes and records in array Q have size of S_Q bytes. Dr. Tuple needs to implement this subroutine with the highest possible performance because it is a hot-spot of the whole data-mining application. However, S_P and S_Q are only known at run-time of application which complicates or makes impossible to make certain well-known compile-time optimizations.

The straightforward way to find byte-offset of i-th record in array P is to use the following formula:

$$\text{Pofs}(i) = S_P \cdot i, \tag{1}$$

and the following formula for array Q:

$$\text{Qofs}(i) = S_Q \cdot i. \tag{2}$$

However, multiplication computes much slower than addition or subtraction in modern processors. Dr. Tuple avoids usage of multiplication while scanning array P by keeping computed byte-offset $\text{Pofs}(i)$ of i-th record instead of its index i in all other data-structures of data-mining application. He uses the following simple formulae when he needs to compute byte-offset of the record that precedes or follows i-th record in array P:

$$\text{Pofs}(i+1) = \text{Pofs}(i) + S_P$$

$$\text{Pofs}(i-1) = \text{Pofs}(i) - S_P$$

Whenever a record from array P is located by either scanning of the array or by taking $\text{Pofs}(i)$ from other data structures, Dr. Tuple needs to retrieve information from the corresponding record in array Q. To access record in array Q its byte-offset $\text{Qofs}(i)$ needs to be computed. One can immediately derive formula to compute $\text{Qofs}(i)$ with known $\text{Pofs}(i)$ from formulae (1) and (2):

$$\text{Qofs}(i) = \text{Pofs}(i)/S_P \cdot S_Q \tag{3}$$

Unfortunately, this formula not only contains multiplication, but also contains division. Even though only integer division is required here, it is still an order of magnitude slower than multiplication on modern processors. If coded this way, its computation is going to consume the most of CPU time in data-mining application for ACM Inc.

After some research Dr. Tuple has discovered that he can replace formula (3) with the following fast formula:

$$\text{Qofs'}(i) = (\text{Pofs}(i) + \text{Pofs}(i) << A) >> B \tag{4}$$

where A and B are non-negative integer numbers, "$<< A$" is left shift by A bits (equivalent to integer multiplication by 2^A), "$>> B$" is right shift by B bits (equivalent to integer division by 2^B).

This formula is an order of magnitude faster than (3) to compute, but it generally cannot always produce the same result as (3) regardless of the choice for values of A and B. It still can be used if one is willing to sacrifice some extra memory.

Conventional layout of array Q in memory (using formula (2)) requires $N \cdot S_Q$ bytes to store the entire array. Dr. Tuple has found that one can always choose such K that if he allocates K bytes of memory for the array Q (where $K \le N \cdot S_Q$) and carefully selects values for A and B, the fast formula (4) will give non-overlapping storage locations for each of the N records of array Q.

Your task is to write a program that finds minimal possible amount of memory K that needs to be allocated for array Q when formula (4) is used. Corresponding values for A and B are also to be found. If multiple pairs of values for A and B give the same minimal amount of memory K, then the pair where A is minimal have to be found, and if there is still several possibilities, the one where B is minimal. You shall assume that integer registers that will be used to compute formula (4) are wide enough so that overflow will never occur.

Input

Input consists of several datasets. Each dataset consists of three integer numbers N, S_P, and S_Q separated by spaces ($1 \le N \le 2^{20}, 1 \le S_P \le 2^{10}, 1 \le S_Q \le 2^{10}$).

Output

For each dataset, write to the output file a single line with three integer numbers K, A, and B separated by spaces.

Sample Input

```
20 3 5
1024 7 1
```

Sample Output

```
119 0 0
1119 2 5
```