The task is to develop a program to plan a military parade.

Assume a military parade is being organized. The parade includes a presentation of military vehicles. The vehicles are presented in a rectangular formation, the formation occupies whole road. That is, there is a vehicle in each lane of the road and all vehicles maintain the same speed. For aesthetic reasons, the formation must be of a rectangular shape, thus the number of vehicles must be a multiple of the traffic lanes. The problem is that there are several sections of the parade, the number of lanes varies among the sections.

The input of the program consists of the road sections, followed by a list of problems to solve. Each road section is described by an integer, the integer denotes the number of traffic lanes in the particular section of the road. This list of integers is enclosed in curly braces, the numbers are separated by commas. There is at least one road section in the input, there is not any explicit limit on the number of road sections. Following the road sections, there is a list of problems to solve. Each problem is described by two integers `from` and `to`. Number `from` denotes the starting position of the military parade, i.e. the index of the first section passed by the vehicles. The second number `to` denotes the end position, i.e. the military parade ends before section `to`. Both indices are zero-based.

The output of the program is the number of vehicles required for the parade that passes sections `from`, `from + 1`, ..., `to - 1`, such that all lanes are occupied and the formation is of a rectangular shape. There is a trivial solution for the problem - 0 vehicles - however that is not acceptable from the propaganda point of view. On the other hand, the number of available vehicles is limited, thus we are interested in the lowest possible number of vehicles to organize the parade.

The program must detect an invalid input. If the input is invalid, the program must display an error message and terminate. The following is considered invalid:

- the number of lanes were not positive integers,
- there were not any road section in the input,
- there are missing/extra commas or braces in the road sections,
- indices `from` or `to` are not integers,
- indices `from` or `to` are out of range (exceed the number of road sections),
- index `from` is greater or equal to index `to`.

The program is tested in a limited environment. Both time and memory is limited. The limits are set such that a correct implementation of a naive algorithm passes all tests. The problem needs to design a reasonable memory representation of the road sections. The available memory is proportional to the input problem size. Therefore, brute force allocation (like statically allocated array of 1000000 elements) will not pass. There is a bonus test included in this homework. The test inputs many road sections and tests a lot of problems for different values of `from` and `to` indices. An efficient algorithm and a preprocessing is required to pass the bonus test.

Sample program run:

---

```
Lanes:
{ 1, 2, 3, 4, 5, 6, 7, 8, 9 }
Routes:
0 9
Vehicles: 2520
0 5
Vehicles: 60
0 6
Vehicles: 60
0 7
Vehicles: 420
0 8
Vehicles: 840
4 7
Vehicles: 210
2 9
Vehicles: 2520
8 9
Vehicles: 9
```

---

```
Lanes:
  {  7  , 3,5 ,11, 8, 16, 4, 9, 2
,8, 4, 2}
Routes:
0 12
Vehicles: 55440
4 6
Vehicles: 16
4 7
Vehicles: 16
4 8
Vehicles: 144
  4    9
Vehicles: 144
4 10
Vehicles: 144
6 5
Invalid input.
```

---

```
Lanes:
{ 1, 2, 3
5 6
Invalid input.
```

---

```
Lanes:
{1,2,3}
Routes:
2 4
Invalid input.
```

```
Lanes:
{ 1, 2, trololo }
Invalid input.
```

---

**Advice:**

- The sample runs above list both the output of your program (bold face text) and user input (regular text). The bold/regular formatting is included here, in the problem statement page, to increase readability of the listing. Your program must output the text without any additional markup.
- The number of lanes in the individual road sections is small, the values fit into `int` data type.
- The resulting number of vehicles may be quite high, `long long int` type is recommended to store the value.
- Textual description of valid input data structure is not 100% exact. Therefore we provide a formal specification of the input language in EBNF:

```
    input      ::= { whiteSpace } '{' { whitespace } integer { lane }
{ whiteSpace } '}'
                   { whiteSpace } { problem } { whiteSpace }
    whiteSpace ::= ' ' | '\t' | '\n' | '\r'
    lane       ::= { whiteSpace } ',' { whiteSpace } integer
    problem    ::= { whiteSpace } integer { whiteSpace } integer
    integer    ::= digit { digit }
    digit      ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```