BITWISE EXPRESSIONS

In signal processing, one is sometimes interested in the maximum value of a certain expression, containing bitwise AND and OR operators, when the variables are integers in certain ranges. You are to write a program that takes such an expression and the range of each variable as input and determines the maximum value that the expression can take.

For simplicity, the expression is of a specific form, namely a number of subexpressions in parentheses separated by the bitwise AND operator (denoted &). Each subexpression consists of one or more variables separated by the bitwise OR operator (denoted |). Using this convention, it is possible to completely specify the expression by giving the number of subexpressions and, for each subexpression, the number of variables in the subexpression. The variables are simply numbered according to their occurrence in the expression.

An example will clarify this. If the number of subexpressions is 4 and the number of variables in each subexpression is 3, 1, 2, and 2, then the expression will be

$$E = (v_1 \mid v_2 \mid v_3) \,\&\, (v_4) \,\&\, (v_5 \mid v_6) \,\&\, (v_7 \mid v_8)$$

The bitwise operators are defined in the common way. For example, to perform the operation 21 & 6, we first write down the binary form of the numbers (operands): 10101 and 110 (since $21 = 2^4 + 2^2 + 2^0$ and $6 = 2^2 + 2^1$). Every binary digit in the result now depends on the corresponding digit in the operands: if it is 1 in *both* operands, the resulting digit will be one, otherwise it will be zero. As is illustrated below to the left, the resulting value is 4. If instead we want to calculate 21 | 6, the procedure is the same except that the resulting digit will be one if the corresponding digit is one in *any* of the operands, and thus it will be zero only in the case that the digit is zero in both operands. As is illustrated below in the center, the result is 23. The generalization to more than two operands is straightforward. The rightmost example below illustrates that 30 & 11 & 7 = 2.

```
  10101        10101          11110
& 00110      | 00110        & 01011
  00100        10111          00111
                              00010
```

INPUT

The input is read from a text file named **`bitwise.in`**. In the first line, two integers N and P are given, where N is the total number of variables ($1 \le N \le 100$) and P is the number of subexpressions ($1 \le P \le N$). In the next line, P integers (K_1, K_2, ... ,K_P) are given, where K_i is the number of variables in the i-th subexpression. The K_i are all greater than or equal to 1 and their sum equals N. Each of the following N lines contains two integers A_j and B_j ($0 \le A_j \le B_j \le 2\ 000\ 000\ 000$), specifying the range of the j-th variable in the expression according to $A_j \le v_j \le B_j$.

OUTPUT

The output is written into a text file named **`bitwise.out`**. The content should be one row with a single integer: the maximum value that the expression can take.

EXAMPLE

Assume that we want to limit the values of the eight variables in the expression above according to $2 \le v_1 \le 4$, $1 \le v_2 \le 4$, $v_3 = 0$, $1 \le v_4 \le 7$, $1 \le v_5 \le 4$, $1 \le v_6 \le 2$, $3 \le v_7 \le 4$, and $2 \le v_8 \le 3$. This corresponds to the following content of **`bitwise.in`**:

```
8 4
3 1 2 2
2 4
1 4
0 0
1 7
1 4
1 2
3 4
2 3
```

If writing in binary notation, one of the best assignments gives the expression (100 | 011 | 000) & (111) & (100 | 010) & (100 | 011), from which we note that all subexpressions may become equal to 7 except the third. Thus, the program should write a file **`bitwise.out`** with the content:

```
6
```

GRADING

In 30% of the test cases, the number of possible assignments is less than one million.

COIN COLLECTOR

In a certain country, there are N denominations of coins in circulation, including the 1-cent coin. Additionally, there's a bill whose value of K cents is known to exceed any of the coins. There's a coin collector who wants to collect a specimen of each denomination of coins. He already has a few coins at home, but currently he only carries one K-cent bill in his wallet. He's in a shop where there are items sold at all prices less than K cents (1 cent, 2 cents, 3 cents, … , $K - 1$ cents). In this shop, the change is given using the following algorithm:

1. Let the amount of change to give be A cents.
2. Find the highest denomination that does not exceed A. (Let it be the B-cent coin.)
3. Give the customer a B-cent coin and reduce A by B.
4. If $A = 0$, then end; otherwise return to step 2.

The coin collector buys one item, paying with his K-cent bill.

Your task is to write a program that determines:

- How many different coins that the collector does not yet have in his collection can he acquire with this transaction?
- What is the most expensive item the store can sell him in the process?

INPUT

The input is read from a text file named **`coins.in`**. The first line of the input file contains the integers N ($1 \le N \le 500\,000$) and K ($2 \le K \le 1\,000\,000\,000$). The following N lines describe the coins in circulation. The $(i + 1)$-th line contains the integers c_i ($1 \le c_i < K$) and d_i, where c_i is the value (in cents) of the coin, and d_i is 1, if the collector already has this coin, or 0, if he does not. The coins are given in the increasing order of values, that is, $c_1 < c_2 < \ldots < c_N$. The first coin is known to be the 1-cent coin, that is, $c_1 = 1$.

OUTPUT

The output is written into a text file named **`coins.out`**. The first line of the output file should contain a single integer — the maximal number of denominations that the collector does not yet have, but could acquire with a single purchase. The second line of the output file should also contain a single integer — the maximal price of the item to buy so that the change given would include the maximal number of new denominations, as declared on the first line.

EXAMPLE

coins.in	coins.out
7 25 1 0 2 0 3 1 5 0 10 0 13 0 20 0	3 6

COUNTRIES

This problem considers how mighty countries arise from humble beginnings. Consider a two-dimensional map of the area. On this map there are n cities. Each city i is in a distinct location with coordinates (x_i, y_i) on the map. The city has also a number s_i of soldiers in it under the command of a general.

The influence city i exerts to another location (x, y) is computed as s_i divided by the squared distance between it and (x, y). It is as if the mass of soldiers in city i exerted a gravitational pull to all map locations around it. City i is threatened by another city j if the influence exerted by city j on the location (x_i, y_i) of city i exceeds its number of soldiers s_i: then city j can dispatch enough soldiers to overpower all the soldiers defending city i. If city i is not threatened by any other city j, then its grateful citizens elect its invincible general as their king, and turn their city into the capital of his kingdom.

On the other hand, if some city j threatening city i exerts strictly more influence on its location (x_i, y_i) than any other city k, then the citizens of city i have no choice: city i must surrender to city j. Henceforth city i must obey the same capital as city j obeys; however, the s_i soldiers in city i do not join the army of city j or the capital. Otherwise city i is saved by mutual distrust of the equally threatening cities j and k: If one of them would attack and overpower city i, then the other would in turn attack and overpower the battle-weary first attackers. However, the citizens of city i can no longer elect its general as their king, because he has failed in his duty to keep the city safe from threats. Thus they must turn their city into the capital of a democracy instead.

Your task is to write a program, which takes the information about the cities on the map as inputs, and outputs for each city i one of the three outcomes:

- It is the capital of a kindgom.
- It is the capital of a democracy.
- It obeys city j as its capital.

INPUT

The input is read from a text file named **`countries.in`**. The first line consists of one integer $1 \le n \le 1000$, the number of cities. The following n lines give the information on the n cities, each city as its own line. Line $i + 1$ gives the information on city i as the three integers x_i, y_i, s_i which are separated by single space characters. All these numbers are in range $0 \le x_i, y_i, s_i \le 1000$.

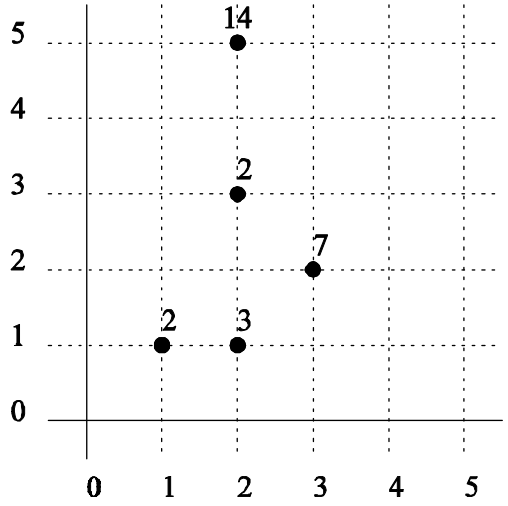
OUTPUT

The output is written into a text file named **`countries.out`**. The output consists of n lines, where line i consists of the outcome for city i:

- The character **`K`** if city i is the capital of a kingdom.
- The character **`D`** if city i is the capital of a democracy.
- The number $1 \leq j \leq n$ of the city which city i obeys as its capital if city i had to surrender.

EXAMPLE

Consider the following map, where each dot represents a city and its number of soldiers is given above it:



Assuming that its 5 cities are numbered from top to bottom and left to right, then the corresponding input and output files are as follows:

`countries.in`

```
5
2 5 14
2 3 2
3 2 7
1 1 2
2 1 3
```

`countries.out`

```
K
D
K
3
3
```

That is, city 3 at location (3,2) is the capital of a kingdom, which also contains the cities 4 at location (1,1) and 5 at location (2,1). On the other hand, city 1 at location (2,5) forms a kingdom all by itself, while city 2 at location (2,3) forms a democracy all by itself.