TASK	NAJBOLJIH 5	OKRET	ZADAĆA	KOMPIĆI	FUNKCIJA	RASPORED
source code	najboljih5.pas najboljih5.c najboljih5.cpp	okret.pas okret.c okret.cpp	zadaca.pas zadaca.c zadaca.cpp	kompici.pas kompici.c kompici.cpp	funkcija.pas funkcija.c funkcija.cpp	raspored.pas raspored.c raspored.cpp
input	standard input (*stdin*)					
output	standard output (*stdout*)					
time limit	1 second	1 second	1 second	1 second	1 second	2 seconds
memory limit	32 MB	32 MB	32 MB	32 MB	128 MB	128 MB
point value	**50**	**80**	**100**	**120**	**150**	**150**
	650					

The Croatian version of this contest has the following rules: "*Each round of the contest consists of 8 tasks with different point values. Every contestant may solve any of the tasks they choose. However, the contestant's final score will be the sum of points earned on any 5 tasks such that this sum is maximized.*"

Since the organizers were busy coming up with interesting problems for the contest (and translating them), they've simply forgotten to solve the problem of determining the points scored by each contestant. Now they are kindly asking you to do it for them.

Write a program that, given the number of points earned by a contestant on each task, determines the total amount of points scored by that contestant, as well as the sorted list of the 5 problems counting towards that score. No contestant will ever score the same amount of points on two different problems.

INPUT

Input consists of 8 lines. Each line of input contains a single positive integer **X** ($0 \leq \mathbf{X} \leq 150$), where the number **X** in row *i* denotes the number of points earned by the contestant on problem *i*. All 8 numbers **X** will be distinct.

OUTPUT

The first line of output must contain the total amount of points scored by the contestant.

The second line of output must contain indices of the 5 problems counting towards the total score, **sorted in ascending order**, separated by single spaces. Problem indices are positive integers from 1 to 8, inclusive.

SCORING

If only the first line of output (the total amount of points) is correct, the solution will be awarded **40% of points** on that test case (even if the second line of output is not present).

SAMPLE TESTS

input	input	input
20 30 50 48 33 66 0 64	20 0 50 80 77 110 56 48	20 30 50 80 110 11 0 85
output	output	output
261 3 4 5 6 8	373 3 4 5 6 7	355 2 3 4 5 8

Mirko has been learning to drive, but he still **cannot make a U-turn in a narrow street**. That's why he has decided to go practice in a town where U-turns are forbidden everywhere. This prohibition can be marked by the following sign:

Mirko has soon figured out that his ideal town must not contain dead-end streets, since it is impossible to exit such a street without a U-turn (let us assume that Mirko cannot drive in reverse either). Write a program to analyse a town map and determine whether the town is suitable for Mirko (i.e. whether the town has any dead-end streets).

The town map is a table with **R** x **C** cells, where each cell is a building segment (denoted by X) or a road surface (denoted by a dot). From a road surface cell, Mirko can move to any of the surrounding four cells (up, down, left, or right), provided that it is also a road surface (i.e. not a building).

Formally, we will determine that a town is free of dead-end streets if, starting from any road surface cell and going in any of the possible directions, we can return to the starting cell without making a 180 degrees turn (changing our direction to the opposite one).

INPUT

The first line of input contains the positive integers **R** and **C** ($3 \leq$ **R**, **C** ≤ 10), the dimensions of the town.

Each of the next **R** lines contains **C** characters, with each character either "X" or ".". These **R** x **C** characters represent the town map as described in the text above. At least two cells will be road surfaces, and all road surfaces will be connected (i.e. mutually reachable).

OUTPUT

The first and only line of output must contain 0 if the town is free of dead-end streets, otherwise it must contain 1.

SAMPLE TESTS

input	input	input
4 3 XXX X.X X.X XXX	5 5 XX.XX X...X ..... X...X XX.XX	3 9 ...XXX... .X.....X. ...XXX...
output	output	output
1	1	0

Mirko has received a homework assignment to compute the **greatest common divisor** of the two positive integers **A** and **B**. Since the numbers are quite large, the teacher provided him with **N** smaller integers whose product is **A**, and **M** integers with product **B**.

Mirko would like to verify his result, so he has asked you to write a program to solve his problem.

If the result is more than 9 digits long, output only the **last 9** digits.

INPUT

The first line of input contains the positive integer **N** ($1 \leq \mathbf{N} \leq 1000$).

The second line of input contains **N** space-separated positive integers less than 1 000 000 000, whose product is the number **A**.

The third line of input contains the positive integer **M** ($1 \leq \mathbf{M} \leq 1000$).

The fourth line of input contains **M** space-separated positive integers less than 1 000 000 000, whose product is the number **B**.

OUTPUT

The first and only line of output must contain the greatest common divisor of numbers **A** and **B**. If the result is more than 9 digits long, output only the last (least significant) 9 digits.

SAMPLE TESTS

input	input	input
3 2 3 5 2 4 5	4 6 2 3 4 1 1	3 358572 83391967 82 3 50229961 1091444 8863
output	output	output
10	1	000012028

First sample description: The greatest common divisor of numbers A = 30 and B = 20 equals 10.

After successfully solving his math homework from the previous task, Mirko has become bored, so he has made a list of **N** large integers. On the list there are some pairs of numbers that he likes, and some pairs he doesn't like.

Mirko has named the pairs that he likes **pals**. Two numbers are **pals** if they have **at least one digit in common** (not necessarily in the same position).

Help Mirko count how many pairs of numbers in his list are pals.

INPUT

The first line of input contains the positive integer **N** ($1 \leq \mathbf{N} \leq 1\,000\,000$).

Each of the next **N** lines contains a positive integer from the range $[1, 10^{18}]$, a number from Mirko's list. No two numbers in the list will be equal.

OUTPUT

The first and only line of output must contain the number of pairs that are pals.

SAMPLE TESTS

input	input
3 4 20 44	4 32 51 123 282
output	output
1	4

Mirko has written the following function:

```
int fun() {
  int ret = 0;
  for (int a = X1; a <= Y1; ++a)
    for (int b = X2; b <= Y2; ++b)
      ...
        for (int <N-th> = XN; <N-th> <= YN; ++<N-th>)
          ret = (ret + 1) % 1000000007;
  return ret;
}
```

```
function fun: longint;
var
  ret: longint;
  a, b, ... , y, z: longint;
begin
  ret := 0;
  for a := X1 to Y1 do
    for b := X2 to Y2 do
      ...
        for <N-th> := XN to YN do
          ret := (ret + 1) mod 1000000007;
  fun := ret;
end;
```

<***N**-th*> denotes the $\mathbf{N}^{\text{th}}$ lowercase letter of the English alphabet. Each ***Xi*** and ***Yi*** denotes either a positive integer less than or equal to 100 000 or a name of a variable that some outer loop iterates over. For example, *X3* can be either a, b, or an integer literal. At least one of ***Xi*** and ***Yi*** will be an integer literal (i.e. not a variable name) for every *i*.

Compute the return value of the function.

INPUT

The first line of input contains the positive integer **N** ($1 \leq \mathbf{N} \leq 26$).

For the next **N** lines, the i^{th} line contains **Xi** and **Yi**, separated with a space. If **Xi** and **Yi** are both integer literals, then **Xi** ≤ **Yi**.

OUTPUT

The first and only line of output must contain the return value of the function.

SAMPLE TESTS

input	input	input
2 1 2 a 3	3 2 3 1 2 1 a	3 1 2 a 3 1 b
output	output	output
5	10	11

Mirko's pizza place is the best one in town. It is so good that all town residents eat pizza for lunch every day. Mirko's delivery service is so fast that the delivery time is negligible. The problem is bakimg the pizzas, since all residents have their own favourite topping combination, so baking pizzas for two different residents doesn't always take the same amount of time. Mirko only has one small baking oven with a capacity of a single pizza at a time, so good scheduling is extremely important and must be determined **before the day starts**.

For each of the **N** town residents (denoted by numbers from 1 to **N**) we know the baking duration for their favourite pizza ($\mathbf{T_i}$), as well as the moment in the day when they plan on having lunch ($\mathbf{L_i}$). If a resident receives their pizza **K** moments before the planned lunch time, Mirko is rewarded with a tip of **K** kunas[1]. On the other hand, if the pizza is delivered **K** moments late (after the planned lunch time), Mirko must pay the resident **K** kunas (because of his timely delivery insurance policy). If the pizza is delivered precisely on time, Mirko won't get a tip, but he doesn't have to pay anything either.

Mirko would like to know the **maximum total tip** (including all insurance payouts as negative tips) that can be earned in a day if pizzas are baked in an optimal order. Notice that Mirko can earn a negative total tip (if he has to pay out more insurance than the amount of tips he receives).

Since residents sometimes change their favourite pizza toppings, as well as their preferred lunch time, Mirko's schedule must be adapted in order to keep earning optimal tip amounts. Write a program to compute the maximum total tip for the beginning requirements, as well as after each change.

Note: In this town, the day starts at the moment $\mathbf{t} = 0$ and lasts much longer than the time needed to bake pizzas for all residents. The schedule, including the adaptations, must be determined before the day starts.

INPUT

The first line of input contains two positive integers **N** and **C**, the number of residents and the number of pizza requirement changes, respectively.

Each of the next **N** lines contains two positive integers: $\mathbf{L_i}$, the moment when resident **i** plans on having lunch, and $\mathbf{T_i}$, the time needed to bake the pizza for resident **i**.

Each of the next **C** lines contains three positive integers: **R** (the index of a resident), **L** (the new moment when resident **R** plans on having lunch), and **T** (the time needed to bake resident **R**'s new favourite pizza).

Constraints:

$1 \leq \mathbf{N}, \mathbf{C} \leq 200\,000$,

$0 \leq \mathbf{L_i}, \mathbf{L} \leq 100\,000$,

$1 \leq \mathbf{T_i}, \mathbf{T} \leq 100\,000$,

$1 \leq \mathbf{R} \leq \mathbf{N}$.

OUTPUT

The first line of output must contain the maximum total tip for the beginning requirements of the residents.

For each of the **C** changes, the output must contain an additional line of output containing the new maximum total tip value after the change.

[1] Kuna - Croatian currency. 1 kuna = 100 lipa(s); 1 € = about 7.5 kuna(s). Come to Croatia! :)

SCORING

In test cases worth 50% of points, the following constraint holds: $0 \leq \mathbf{T_i}, \mathbf{T} \leq 1000$.

SAMPLE TESTS

input	input	input
3 2 10 2 6 5 4 3 1 6 1 3 0 10	4 2 3 2 0 3 4 3 4 1 3 0 4 1 4 5	6 7 17 5 26 4 5 5 12 4 8 1 18 2 3 31 3 4 11 5 4 19 3 5 23 2 6 15 1 5 19 1 3 10 4
output	output	output
3 2 -11	-8 -13 -18	27 59 56 69 78 81 82 58

First sample description: The optimal pizza baking schedule is (1, 3, 2). That way, the first pizza will be finished at the moment t = 2, the third one at t = 5, and the second one at t = 10. The first pizza will be delivered 8 moments early (8 kunas tip), the second one will be 1 moment late (-1 kuna), and the third one will be 4 moments late (-4 kunas), so the total tip is 3 kunas. When the first resident changes requirements, the optimal schedule remains unchanged, while the tips change to 5, 0, and -3, respectively. After the second requirement change, the optimal schedule is (1, 2, 3), while the tips are 5, 0, and -16, respectively.