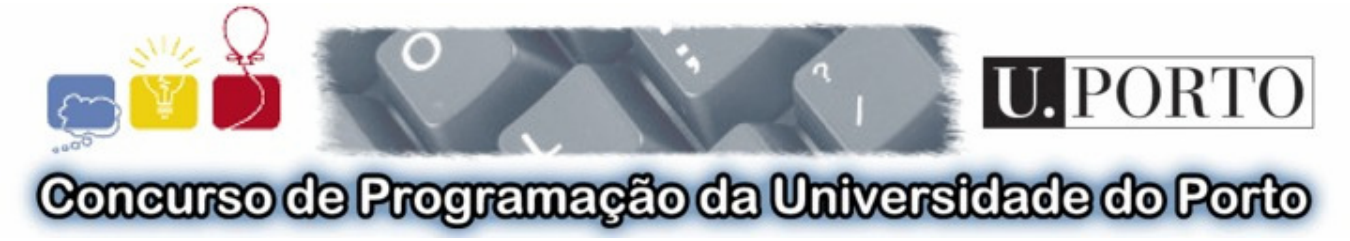


3ª Prova Local

U.PORTO

10 de Outubro de 2007
16:00 às 20:00
7 problemas

(página intencionalmente deixada em branco)

Problem A - Blocks are falling

Blocks are vertically falling... some stay over the floor; others stay over previous blocks, depending on their horizontal position and dimensions. All blocks are one unit height and maintain their horizontal position and orientation.

Figure 1 presents an example. The identifier of a block is related to the order of falling: block 1 is the first to fall, block 2 is the second, and so on.

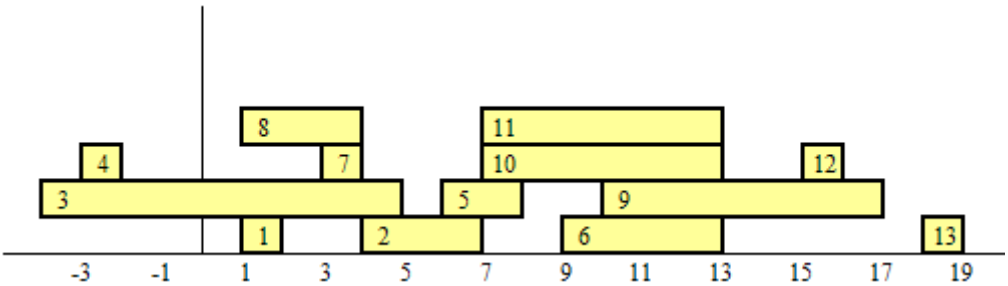


Figure 1: Example of 13 blocks falling

Since blocks 1 and 2 do not intersect, they stay over the floor or, in other terms, they stay on level 1; however, block 3 has to stay over the above, so it remains on level 2 and, when block 4 falls, it stays on level 3. Similar rules can be applied to the resting blocks.

The Problem

Given a sequence of falling blocks, the problem is to determine the maximum level attained and identify the correspondent blocks. In the example given, it can be seen that the result is 4, with blocks 8 and 11. Each block is described by its left coordinate x (any decimal value, positive or negative) and its length (any integer value larger than zero).

Input

Input is given as a list of text lines, the first containing the number ***N*** of blocks for the problem.

It is followed by ***N*** lines containing a coordinate ***x*** (decimal value) and a length (integer value) separated by one or more spaces. Identifier of the corresponding block is implicit in these lines sequence (block 1, 2, etc.).

The number of blocks is limited to 1000. It is ensured that no coincidences can occur in coordinates ***x*** of right side and left side of contiguous blocks.

Output

One text line containing the level result and another line containing the ***M*** number of blocks in that situation. These are followed by ***M*** lines containing the identifiers of the ***M*** blocks found in that level, in the order left to right.

Sample Input

```
13
1.048 1
4.091 3
-3.919 9
-2.984 1
6.054 2
9.031 4
3.09 1
1.047 3
10.038 7
7.097 6
7.014 6
15.078 1
18.03 1
```

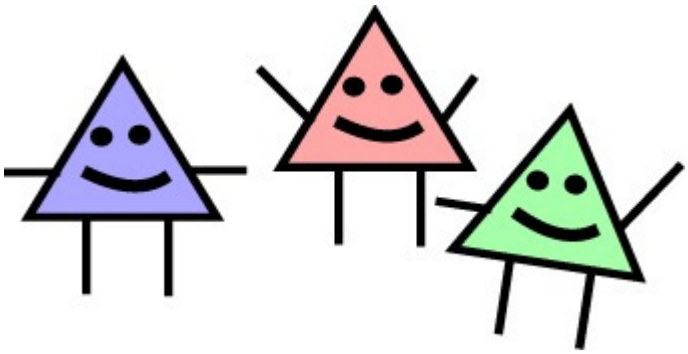
Sample Output

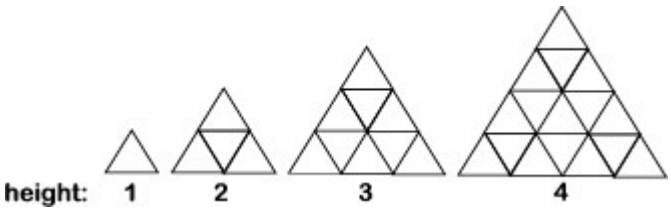
```
4
2
8
11
```

Note: the sample input and output correspond to the example given in figure 1.

Problem B - Triangle World

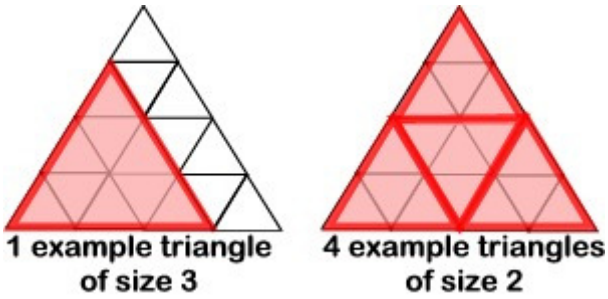
Far, far away there exists a world where everything is a triangle. Even its living forms have the shape of a triangle. The beings, the planet's inhabitants, are called "trianglers". They completely worship triangles, and everything related to them. Naturally, the trianglers don't have usual square grids as we have, since they only use triangles. They use a kind of triangular grid, as we can see in the following figure.





The triangle grids of triangle world.

A triangle grid can be identified by the height of the smaller triangles, as you can see in the figure. The trianglers need to know how many triangles do the grid points form. Note that the triangles can be of different sizes. For example, in a grid of height 4, we can find 16 small triangles of size 1, but we can also find others with size 2, 3 or 4. The following figure shows some example of triangles of size 2 and 3 that we can find in a grid of height 4. But can you help the trianglers discover the total number of triangles on a grid?



Some examples of triangles found on a grid of height 4

The Problem

Given a grid of a determined height, you have to calculate the total number of different triangles that can be formed with the points of that same grid.

Input

The first line of input will contain an integer number **C**, indicating the number of cases that follow ($1 \leq C \leq 100$).

The following *C* lines will contain each one a single integer number **H**, indicating the height of the triangle grid to consider in that case ($1 \leq H \leq 1000$).

Output

For each input case you must output a single line, in the format `"Height H: NUMBER_OF_TRIANGLES"`, indicating the height of the triangle grid and the respective total number of triangles that can be found on that grid. You can be assured that the number of triangles will be smaller than 2^{31} and therefore will fit on a normal signed 4-byte integer.

Sample Input

```
3
1
2
3
```

Sample Output

```
Height 1: 1
Height 2: 5
Height 3: 13
```

Problem C - Flamel Goes to Spain

> *Q:* *When the Philosophers speak of gold and silver, from which they extract their matter, are we to suppose that they refer to the vulgar gold and silver?*
>
> *A:* *By no means; vulgar silver and gold are dead, while those of the Philosophers are full of life.*

(Alchemical Catechism, Paracelsus, Sec. XVI)

Nicolas Flamel was a famous alchemist born in Paris in 1330. Flamel made it his life's work to understand the text of a mysterious twenty-one-page book he had purchased and that has been identified by a sage as the *Book of Abraham* also known as the *Codex*. Around 1378 he traveled to Spain for assistance with translation, but before hitting the road he had a problem to solve.

No alchemist would travel without their vast collection of flasks and vials, all of them full of mysterious substances of extreme importance for their work. Flamel was no exception. He wanted to carry as much substances as possible but he didn't have enough containers to bring all of them. Traveling with this kind of substances required some specially designed containers.

But Flamel had a plan. He made a list of all the substances he could not live without. This list included the quantity needed of each substance and also the quantity he had stored. Then he found out the price of each substance in Spain. His idea was to take with him larger quantities of each substance he needed and then sell the excess and buy those he didn't bring. This was an incredible smart plan as some of his containers were much larger than he needed but he could not mix two different potions in the same container.

The Problem

Given a list of substances and containers, discover the best way to accommodate the substances in the containers in order to make a profit when selling the substances in excess. This profit should be enough to buy substances not carried by Flamel. As the price of substances in Spain was much higher than in France he decided to make the maximum profit possible.

For instance, if Flamel had 3 litters of *ammonia*, 2.5 litters of *alcohol* and 0.5 litters of *formic acid*, and he needed to carry at least 0.5 litters of each of these substances in one single container with a 5 litter capacity, he wouldn't be able to do so because he coulnd't mix substances.

But if he knew he could sell *ammonia* in Spain for 23 *maravedis/l* (the *maravedis* was one of Spanish many currencies at the time) and buy alcohol for 10 *maravedis/l* and *formic acid* for 15 *maravedis/l*, he could bring 3 litters of *ammonia*, sell 2.5 litters for

57.5 *maravedis* and use 5 *maravedis* to buy 0.5 litters of *alcohol* and 7.5 *maravedis* to buy 0.5 litters of *formic acid*. He would even make a profit of 45 *maravedis*.

Input

The first line of the input contains a single integer *p* representing the number of different potions possessed by Flamel . This line will be followed by *p* lines each one of them describing a substance. Each of these lines will contain two real numbers *qt* (the quantity Flamel has of the substance) and *mn* (the minimum quantity of the substance Flamel must have when in Spain) and one integer *sp* (the price of the substance in Spain in *maravedis*).

The following will always be true: 1 <= *p, qt, mn* <= 100 ; *mn* <= *qt* and 1 <= *sp* <= 1000.

The next line will contain two integers: *n* (1 <= n <= 100) representing the number of containers Flamel possesses and *c* (1 <= c <= 50) the capacity of each container in litters.

Output

The output should contain a single line, with a real number, precise to the 3rd decimal place, representing the profit Flamel can make with his plan, or the world *impossible* if it isn't possible to get the minimum quantities of each substance he desires.

Example Input 1

```
3
3 0.5 23
2.5 0.5 10
0.5 0.5 15
1 5
```

Example Output 1

```
45.000
```

Example Input 2

```
3
3 0.5 23
2.5 0.5 10
0.5 0.5 15
1 1
```

Example Output 2

```
impossible
```

Problem D - Entering the Stadium

Every time you go to a football game, you cannot stop to notice that huge crowds do the same and long lines form to enter the stadium. The stewards have to investigate everyone, searching for forbidden objects, and then the fans have to show their tickets to the digital readers, to confirm that they really have a valid ticket and are allowed to enter. All this process causes delays and football fans have to come early to the stadium to see the game.

The presidents of football clubs all want to give their club's associates the best experience, and they want to minimize the time people spend in the lines to enter the stadium. They can for example increase the number of doors and stewards to augment the flow of people entering. But of course this costs money, and they always want to save on this important aspect. But what is the right balance? Can you help to discover the right balance of costs, by finding the minimum number of stewards and ticket readers to provide a determined quality of service in the entrance of the stadium?

You will consider a simplified version of the entrance on a stadium. Suppose you will only consider a specific sector of the stadium, which has **K** doors. Each one of these doors has a single steward, ticket reader and a separate line of persons forming. For simplicity the lines can be numbered from 1 to K. Each time a fan arrives, he chooses the line with the minimum amount of persons waiting to enter (in case of a tie, he chooses the line wth the smallest number). When someone is on the front of the line, it is searched by a steward and shows the ticket for a determined number of seconds, which is constant for every line. After that, the person can enter the stadium. The time a fan waits on the line is the time it takes from the arrival to one of the lines until it can enter the stadium after showing the ticket.

For example, imagine you have 3 doors and that the time it takes to search and show the ticket is 10s. Consider that 6 fans arrive at 3, 6, 7, 11, 17 and seconds after the initial opening of the doors. The first fan goes to line 1 and immediately is searched and shows the ticket. With the first fan still waiting, the second fan arrives, and seeing one person on line 1 (still with the steward), it opts for line 2, with zero fans waiting (using the rule described before). Then comes fan nr.3 at 7 seconds, and opts for line 3. After that comes fan nr 4, which goes to line 1, followed by fan nr 5, which goes to line 2. Then, at 13s, the first fan enters the stadium, and fan nr 4 goes to the steward of line 1. At 16s, fan nr 2 enters the stadium and fan nr 5 goes to the front of that line. At 17s, fan nr 3 enters the stadium, leaving line 3 empty, At the same time, fan nr 6 arrives, and seeing

line 3 empty (the others have one fan) it goes to line 3. Then fan nr 4 enters the stadium at 23s, fan nr 5 enters at 26 and fan nr 6 enters at 27. The fan which waits more time is fan nr 5, since he waits 14s since his arrival to his entrance.

With this scheme, could you help the presidents discover the minimum amount of doors you need to ensure that the fan that waits more, does not wait more than a determined amount of time?

The Problem

You will be given the time it takes to search and show a ticket, and also the arrival times of several fans. Your task is to calculate the minimum number of doors needed to ensure that the maximum amount of time a fan waits to enter the stadium is not bigger than a specific limit.

Input

The first line of input contains two space-separated integers **T** and **L**, where **T** indicates the time it takes a fan to be searched by a steward (and then show the ticket to the reader), and **L** indicates the maximum amount of time you want a fan to wait ($1 \leq T \leq 1000$, $1 \leq L \leq 500000$ and $T \leq L$).

The second line of input contains a single integer **F**, indicating the number of fans to consider ($1 \leq F \leq 150$). Then follow exactly *F* lines, each one with a single integer $\mathbf{F_i}$, indicating the arrival time of fan number i ($1 \leq F_i < 2^{31}$). You can be assured that these times will come in increasing order, and that no two fans arrive at the same time.

Output

The output should contain a single line in the format **"K M"**, where **K** indicates the minimum number of doors needed to ensure that the maximum time a fan waits is smaller or equal than **L**, and **M** is the maximum time that a fan waits in that case (with *K* doors).

Sample Input

```
10 20
6
3
6
7
11
12
17
```

Sample Output

```
3 14
```

Problem E - Reforestation Plan

Deforestation is a major environmental concern for all of us. There can be many causes that lead to deforestation, some are natural causes such us forest degradation or wildfires, but many other causes can the result of direct or indirect human intervention, for example sudden clearcutting or slash-and-burn for agriculture and grazing animals, urban development, acid rain and deliberate fires. Deforestation of tropical rainforest has attract most attention, but the problem is far worse on other forests. Naturally, this leads to loss of biodiversity.

Many governments have started to pay attention to this problem and have also understood that forest management with long-term plans not only can reverse deforestation but also be an economic asset.

The Portuguese government has also a reforestation plan to recover its forests. It was conceived taking into consideration many factors, thus achieving an internal representation plan. The plan, however, is not in an easy readable format to be used by the tree plantation workers.

A reforestation plan is represented by placing numbers on a grid drawn on top of a region map. A number on a grid-cell represents the number of trees that are adjacent (horizontally, vertically and diagonally). Grid-cells with numbers cannot have a tree in them and a tree cannot be adjacent (in all directions) to any other tree.

You have been invited to produce a readable representation of the reforestation plan. The figure below illustrates a plan in its internal representation and one corresponding readable format ('T' means a tree):

	2	1	2	
1				
1		3		
				1

-->

T				T
		T		
	T		T	

Internal Representation Readable Representation

Problem

Given a plan in its internal representation, your task is to print the corresponding readable plan. A plan is a matrix *NxM* of integers, greater or equal zero, where *N* and *M* are the number of rows and columns. To avoid multiple solutions, you should decide to build the readable map with the minimum possible number of trees. In case there are several one respecting this criteria, youl should choose the one that allocates trees to the top-left as most as possible, that is, if you concatenate all lines of the grid, the one which would be lexicographically smaller, considering that a tree would be "smaller" than an empty space. You can see two examples of this ordering in the figure below:

```
T..T                .T.T              .T.T                .T..
.... is smaller then ....      and   .... is smaller than ...T
T..T                T..T              ...T                T...
```

Input

The first line of input includes two numbers *N* and *M* (with *1<N,M<10*) that define the grid for the map. The next *N* lines include a sequence of *M* of integers, greater or equal zero. A -1 value indicates a cell without information. A zero or a positive number, indicates the number of trees that are adjacent to that cell.

Output

The output must be the readable grid map. A tree should be written as 'T' and a '.' represents an "empty cell". If there several possible readable representations, choose the one that respects the disambiguation criteria defined above.

You can assume that the input will always be valid and that at least one readable representation of it will be possible.

Sample Input

```
4 5
-1 2 1 2 -1
1 -1 -1 -1 -1
1 -1 3 -1 -1
-1 -1 -1 -1 1
```

Sample Output

```
T...T
..T..
.....
.T.T.
```

Problem F - Sudokube

The Rubik's Cube, invented in 1974 by the Hungarian Ernõ Rubik, is probably one of the best known puzzles in the world. Sudoku, a number puzzle invented in 1979 is another known puzzle that became famous in the present days after being a huge success in Japan.

It did not take a long time before someone came up with the idea of joining the main concepts of the two puzzles in one single challenge. Instead of colors, the cube has numbers, and the objective is to make each face of the cube have all different digits, as in each square of the normal Sudoku puzzle. They gave it the name of **Sudokube**.

For the purpose of this problem you will have your task simplified by only having to consider a 2x2x2 cube. It has digits in the range [1..4] and the objective is to go from a given configuration to a goal state, where each 2x2 cube face has all 4 different digits, without repetitions. Note that since every digit appears 6 times and that a single face can have the four numbers in many different forms, there will be more than one possible goal state, but every one of them is acceptable, as long as it respects the pre-defined condition of having every face with all the numbers.

As in the case of Rubik's cube, you can rotate 90 degrees a single face of the cube, which affects not only that face, but also the ones which are connected to it. We call a "move" to a single 90 degrees rotation of one cube face (note that you can rotate to both sides).

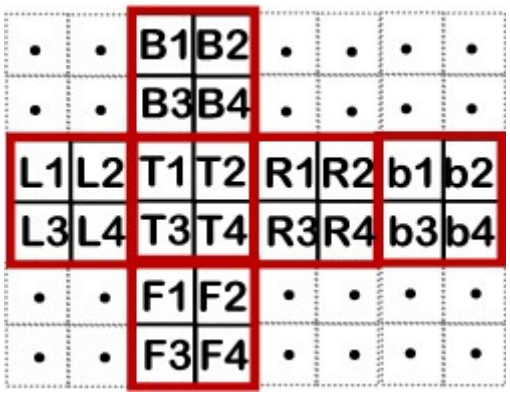
Can you solve this simplified Sudokube?

The Problem

Given a configuration of a 2x2x2 Sudokube, your task is to discover the minimum number of moves to reach a goal state, where each cube face has all the 4 digits from 1 to 4, and no digit is repeated on the same face.

Input

The input will consist of 6 lines representing the configuration of a 2x2x2 Sudokube. Each line will contain 8 characters ('.' or a digit from 1 to 4), as explained on the following figure:



Description of input

In the figure, 'T' represents the top of the cube, 'F' means front, 'B' means back, 'L' means right, 'R' means right and 'b' means bottom. You can imagine this representation as a foldable model of the 3D cube. Just as an example, if you rotate the top face to the left, and considering the previous figure, the resulting Sudokube would become:

```
  .   . B1 B2  .  .  .  .
  .   . R1 R3  .  .  .  .
 LX  B4 T2 T4 F2 RX b1 b2
 LX  B3 T1 T3 F1 RX b3 b4
  .   . L2 L4  .  .  .  .
  .   . FX FX  .  .  .  .
```

See the sample input to better understand. You can be assured that the Sudokube will always be valid, that is, it will only contain digits from 1 to 4, and that it will be solvable, in the sense that it always exist a sequence of moves leading to a goal state. More than that, the Sudokube will be solvable in 5 moves at most.

Output

The output should contain a single integer indicating the minimum number of moves (90 degrees rotations of a single face) needed to reach a goal state from the given configuration. If the input is already a goal state you should print a zero (0), meaning no moves are needed.

Sample Input

```
..21....
..12....
31212412
13434234
..34....
..43....
```

Sample Output

```
2
```

Problem G - Summing Numbers

With random thoughts running trough your mind, you could not stop noticing something special about the sequence of numbers you had written:

```
6 2 1 4
```

The special thing you notice was that by adding a contiguous set of numbers from this sequence you could obtain all numbers from 1 to 9! See how you could do this(the numbers to sum are marked with '^'):

```
6 2 1 4 | 6 2 1 4 | 6 2 1 4 | 6 2
1 4 | 6 2 1 4 |
    ^       ^         ^ ^           ^       ^ ^

6 2 1 4 | 6 2 1 4 | 6 2 1 4 | 6 2 1 4
^           ^ ^ ^   ^ ^       ^ ^ ^
```

You became curious about this property, and decided to create a program to help you discover sequences like this. However, you would like to consider a more general case. You want sequences of size **N**, which have **numbers (not digits)** bigger or equal than **K** that have sums of contiguous numbers in the sequence that match all numbers from **S** to **E**. Knowing N, K and S, you want to discover the sequence that maximizes E, that is, the one that creates the biggest consecutive sequence of numbers.

For example, if N=4, K=1 and S=1, the sequence above would give E=9, (since it produces the numbers from 1 to 9), and in that case 9 is also the maximum possible E. Could you solve the general case?

The Problem

Given three integers **N**, **K** and **S**, you have to find all sequences of N numbers bigger or equal to K that maximize E, given that all integer numbers in the range S to E (inclusive) can be obtained by adding contiguous numbers of the sequence.

Input

The input will be formed by a single line with three integers **N**, **K** and **S** separated by single spaces ($1 \leq N,K,S < 7$ and $K \leq S$). These integers indicate respectively the size of the sequence to consider, the minimum value of a number in the sequence and the start number to which we should discover consecutive contiguous sums from the sequence.

Output

The first line of output should contain a single integer **E**, indicating the maximum possible E, such that the range S to E (inclusive) can be constructed from the sequence as defined above.

The second line of output should contain a single integer **NUM**, indicating that there are exactly NUM different sequences which generate all numbers from S to E. Then follow exactly NUM lines, each one with N integers indicating the maximizing sequence. The numbers should be separated by single spaces, with no leading or trailing spaces. These sequences should also come in lexicographical order.

Sample Input 1

```
4 1 1
```

Sample Output 1

```
9
8
1 1 4 3
1 3 3 2
2 3 3 1
2 5 1 3
3 1 5 2
3 4 1 1
4 1 2 6
6 2 1 4
```

Sample Input 2

```
4 2 5
```

Sample Output 2

```
10
4
2 5 3 6
5 2 2 6
6 2 2 5
6 3 5 2
```