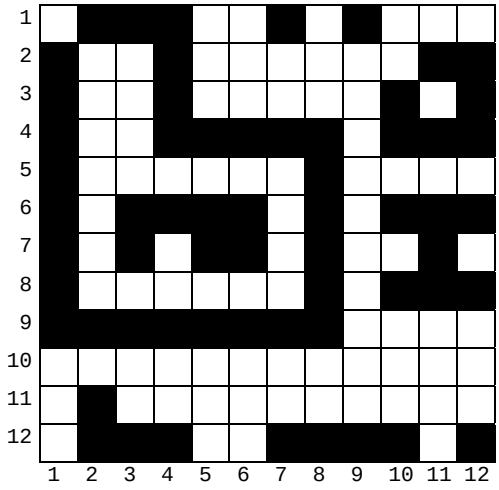
The board of "Ships" game consists of N*N squares. Each square may belong to some ship or be empty. If two squares belong to ships and have common edge, then both squares belong to the same ship. Squares of different ships have no common points. *Tonnage* of ship is number of squares belonging to this ship.

In the given example squares belonging to ships are marked black and on the game board there are: one 29-ton ship, three 7-ton ships, two 4-ton ships and three one-ton ships.



Task

Write program which for given description of game board calculates number of ships and tonnage of each ship.

Input

In the first line of text file `ships.in` one positive integer N (N<30000) is given.

In each of next N input file lines there is given information about one board row, consecutively describing groups of squares from left to right, which belongs to ships in one of two formats:

- `<number_of_square_column>`
  , if square in given column belongs to ship, but squares to the left and to the right are free;
- `<number_of_first_square_column>-<number_of_last_square_column>`
  , if all consecutive squares from first to last (including) belongs to ship and squares to the left and to the right from this group are free.

Square groups are separated by commas, each line ends with semicolon. There are no spaces in lines. If in some board row there are no ship's squares, then corresponding file line contains only one semicolon. It is known that total number of ships does not exceed 1000 and tonnage of any ship does not exceed 1000 tons.

Output

In the text file `ships.out` your program must output information about ships. In each file line there must be exactly two integers separated by space symbol. First number must be tonnage and second must be number of ships having this tonnage. Tonnages must be given in decreasing order and tonnage must be outputed only if there is at least one ship having this tonnage.

Example (corresponds to given example)

ships.in	ships.out
12 2-4,7,9; 1,4,11-12; 1,4,10,12; 1,4-8,10-12; 1,8; 1,3-6,8,10-12; 1,3,5-6,8,11; 1,8,10-12; 1-8; ; 2; 2-4,7-10,12;	29 1 7 3 4 2 1 3

SCALES

You are given an equal arm scales, a set of weight pieces and an object. The pieces are of weight 1,3,9,27,81,..., i.e. the weight of each piece is a power of 3, and for each integer $k \geq 0$ there is exactly one piece of weight 3^k. The object's weight is m, where m is a positive integer. Your task is to put the object on the left scale pan and to put some pieces on one or both scale pans, so that the scales is in balance.

Task

Write a program that:

- reads the object's weight m from the text file `scales.in`,
- calculates which pieces should be put on the left and right scalepan,
- writes the results to the text file `scales.out`.

Input

The first line of the input file `scales.in` contains one integer m, $1 \leq m \leq 10^{100}$

Output

The output file `scales.out` should consist of two lines.

The first line should contain information about pieces put on the left scale pan. First number must be non-negative integer - number of pieces put on the left scale pan followed by weights of pieces in increasing order. Numbers must be separated by single spaces.

The second line must contain information about pieces put on the right scale pan in the same format as first line.

Examples

scales.in	scales.out
42	3 3 9 27 1 81

scales.in	scales.out
30	0 2 3 27

Short formulation. The number sequence is given. Your task is to construct the increasing sequence that approximates the given one in the best way. The best approximating sequence is the sequence with the least total deviation from the given sequence.

More precisely. Let $t_1, t_2, \ldots, t_N$ is the given number sequence. Your task is to construct the increasing number sequence $z_1 < z_2 < \ldots < z_N$.

The sum $|t_1 - z_1| + |t_2 - z_2| + \ldots + |t_N - z_N|$ should be a minimal feasible.

Input

There is the integer N ($1 \le N \le 1000000$) in the first line of input file `seq.in`. Each of the next N lines contains single integer – the given sequence element. There is t_K in the (K+1)-th line. Any element is satisfying to relation $0 \le t_K \le 2000000000$.

Output

The first line of output file `seq.out` must contain the single integer – the minimal possible total deviation. Each of the next N lines must contain single integer – the recurrent element of the best approximating sequence.

If there are several solutions, your program must output any one sequence with a least total deviation.

Example

seq.in	seq.out
7 9 4 8 20 14 15 18	13 6 7 8 13 14 15 18