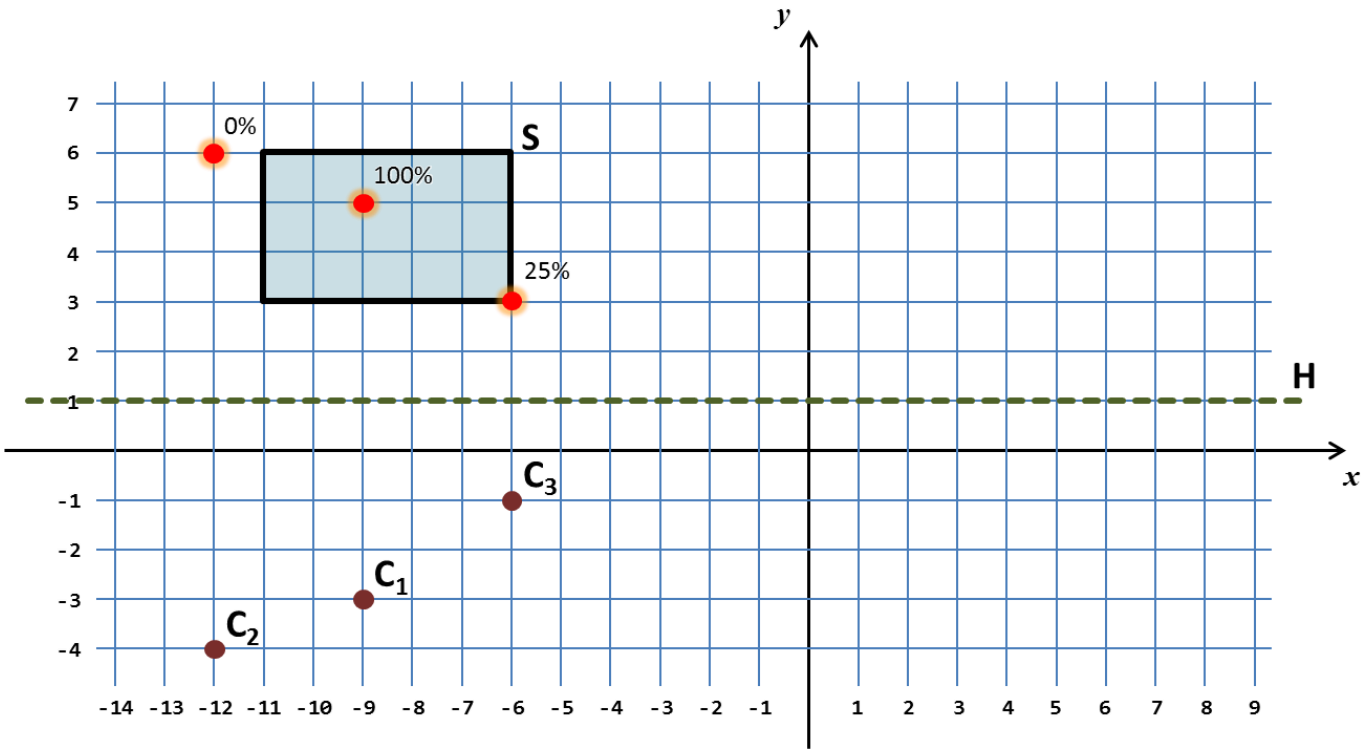
Telerik Software Academy – C# Fundamentals Part 1 – Exam

Problem 1 – Ship Damage

Inside the sea (a standard Cartesian /rectangular/ coordinate system) we are given a ship **S** (a rectangle whose sides are parallel to the coordinate axes), a horizontal line **H** (the horizon) and three catapults, given as coordinates $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ that will be used to fire the ship. When the attack starts, each catapult shoots a projectile exactly into the positions that are symmetrical to $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ with respect to the horizon **H**. When a projectile hits some of the corners of the ship, it causes a damage of 25%, when it hits some of the sides of the ship, the damage caused is 50% and when it hits the internal body of the ship, the damage is 100%. When the projectile does not reach the ship, there is no damage. The total damage is a sum of the separate damages and can exceed 100%.

At the figure below a sea, a ship **S**, a line **H**, three points $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ and their hit positions are shown:



Your task is to write a program that calculates the total damage caused after the attack over the ship.

Input

The input data should be read from the console.

There will be exactly **11** lines holding the integer numbers $\mathbf{S_{X1}}$, $\mathbf{S_{Y1}}$, $\mathbf{S_{X2}}$, $\mathbf{S_{Y2}}$, **H**, $\mathbf{C_{X1}}$, $\mathbf{C_{Y1}}$, $\mathbf{C_{X2}}$, $\mathbf{C_{Y2}}$, $\mathbf{C_{X3}}$, and $\mathbf{C_{Y3}}$.

The ship **S** is given by any two of its opposite corners and is non-empty (has positive width and height).

The line **H** is given by its vertical offset.

The points $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ are given as couples of coordinates and cannot overlap each other.

The line and the ship cannot overlap.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

The output should consist of a single line holding the total damage given as percentage.

Constraints

- The numbers S_{X1}, S_{Y1}, S_{X2}, S_{Y2}, **H**, C_{X1}, C_{Y1}, C_{X2}, C_{Y2}, C_{X3}, and C_{Y3} are all integers between -100 000 and 100 000, inclusive.
- Allowed work time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Input	Output
-11 6 -6 3 1 -9 -3 -12 -4 -6 -1	125%

Input	Output
-6 6 -11 3 1 -9 -4 -11 -1 2 2	75%

Problem 2 – Tribonacci

The Tribonacci sequence is a sequence in which every next element is made by the sum of the previous three elements from the sequence.

$$T_n = T_{n-1} + T_{n-2} + T_{n-3}$$

Write a computer program that finds the **N**th element of the Tribonacci sequence, if you are given the first three elements of the sequence and the number **N**. Mathematically said: with given T_1, T_2 and T_3 – you must find T_n.

Input

The input data should be read from the console.

The values of the first three Tribonacci elements will be given on the first three input lines.

The number **N** will be on the fourth line. This is the number of the consecutive element of the sequence that must be found by your program.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

At the only output line you must print the **N**th element of the given Tribonacci sequence.

Constraints

- The values of the first three elements of the sequence will be integers between -2 000 000 000 and 2 000 000 000.
- The number **N** will be a positive integer between 1 and 15 000, inclusive.
- Allowed working time for your program: 0.25 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
1 1 1 4	3
2 3 4 10	335

Problem 3 – Fir Tree

Christmas Eve is coming so even programmers have to prepare!

In the spirit of the event your task is to write a program that prints a fir tree to the console.

The format of the tree is shown in the examples bellow.

Input

The input data should be read from the console.

On the only input line you have an integer number **N**, showing the height of the tree.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

You must print the fir tree on the console. Each row contains only characters "." (point) or "*" (asterisk).

The first row should have exactly one "*" in the middle (that is the top of the tree) and each of the next lines two more.

The last line should have exactly one asterisk in the middle, showing the stem of the tree.

Constraints

- The number **N** is a positive integer between 4 and 100, inclusive.
- Allowed working time for your program: 0.25 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
5	...*... ..***.. .*****. ******* ...*...
9	........*........ ....***....... .....*****..... ....*******.... ...*********... ..***********.. .*************. *************** ....*........

Problem 4 – We All Love Bits!

One of the things the programmers love the most is bitwise operations. The "bitwise guy" is a synonym for a programmer that loves bits more than everything else in programming. Mitko is a "bitwise guy". He invented a new bitwise algorithm. The algorithm takes one positive integer **P**, makes magic with it and returns a new positive integer. He also defined a new number **P̃** which represents the number **P** in binary numeral system with inverted bits. All zeros in **P** are ones in **P̃** and all ones in **P** are zeros in **P̃**. For example if we have P = 9 (which is 1001 in binary numeral system) its inverted number P̃ will be equal to 6 (which is 110 in binary numeral system). But that's not all! He invented another number **P̈**, which represents reversed number **P** in binary numeral system. For example if we have P = 11 (which is 1011 in binary numeral system) its reversed number P̈ is equal to 13 (which is 1101 in binary numeral system). Mitko's magical algorithm takes a number **P** and transforms it to a new number $\mathbf{P_{new}}$ using the following bitwise transformation: $\mathbf{P_{new} = (P \wedge \tilde{P}) \& \ddot{P}}$.

Your task is to write a program that transforms a sequence of **N** positive integer numbers using Mitko's algorithm.

Input

The input data should be read from the console.

At the first input line there will be one positive integer – the number **N**.

At each of the next **N** lines there will be one positive integer – the consequent number that must be converted using Mitko's algorithm.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

The output must consist of **N** lines, containing the transformed numbers for each number from the input.

Constraints

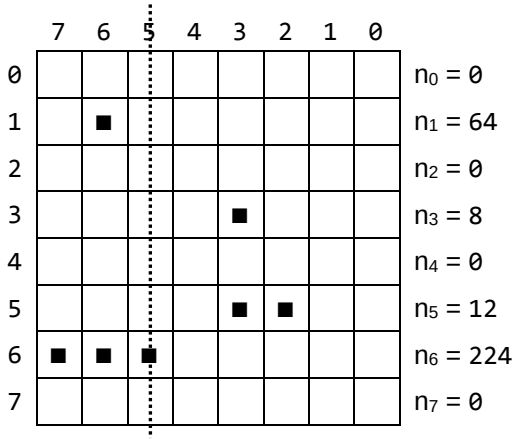
- The number **N** will be positive integer number between 1 and 800, inclusive.
- Each of the **N** numbers will be positive integer numbers between 1 and 2 147 483 647, inclusive.
- Allowed working time for your program: 0.20 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
1 2	1
2 19 248	25 31

Problem 5 – Pillars

You are given a list of **8** bytes (positive integers in the range [0…255]) $\mathbf{n_0}$, $\mathbf{n_1}$, …, $\mathbf{n_7}$. These numbers represent a square grid consisting of **8** lines and **8** columns. Each cell of the grid could either be empty or full. The first line is represented by the bits of $\mathbf{n_0}$, the second – by the bits of $\mathbf{n_1}$ and so on, and the last line is represented by the bits of $\mathbf{n_7}$. Each bit with value 1 denotes a full cell and each bit with value 0 denotes an empty cell. The lines are numbered from the first (top) to the last (bottom) with the numbers 0, 1, …, 7. The columns are numbered from right to left with the indices 0, 1, …, 7. The figure shows a square grid and its representation by a sequence of 8 numbers $\mathbf{n_0}$, $\mathbf{n_1}$, …, $\mathbf{n_7}$:



We are allowed to put a vertical pillar over any of the columns in the grid. Pillars split the grid into two sides (left and right) and the column holding the pillar is ignored. Write a program that finds the leftmost column where the pillar can be put so that the full cells on the left side and on the right side are equal number. For example, in the figure if we put the pillar at column 5, it will split the grid into two sides and both sides will have exactly 3 full cells.

Input

The input data should be read from the console.

There will be exactly 8 lines each holding the integer numbers $\mathbf{n_0}$, $\mathbf{n_1}$, …, $\mathbf{n_7}$.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

If a pillar splitting the grid into two vertical sides each holding the same number of full cells exists, print its column index on the first line and the number of full cells in each of the sides. If multiple pillars can do the job, print only the leftmost. If no such pillar exists, print the string "No" on the console (just one line holding the word "No").

Constraints

- The numbers $\mathbf{n_0}$, $\mathbf{n_1}$, …, $\mathbf{n_7}$ are positive integers in the range [0…255].
- Allowed work time for your program: 0.25 seconds.
- Allowed memory: 16 MB.

Examples

Input Example	Output Example
0 64 0 8 0 12 224 0	5 3
3 0 0 0 0 0 0 0	No