Problem A. A+B

Input file:	`aplusb.in`
Output file:	`aplusb.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

The *Research Institute of Given Strings* (*RIGS*) is a well-known place where people investigate anything about strings. Peter works in the department of string operations of *RIGS*. His department invents different ways to add, multiply, divide strings and even to take logarithm of string based on another one.

Peter's task is to invent an operation of *sum* of two given strings. The current Peter's proposal allows to add only strings of the same length n. To do this, Peter takes the set of all strings of length n over a fixed alphabet. Then Peter sorts all taken strings, thus obtaining a sorted sequence of strings T. Let's denote the length of T as M, and enumerate the elements of this sequence as T_0, $T_1 \ldots T_{M-1}$. Now Peter says that the sum of two strings $A = T_a$ and $B = T_b$ is a string $C = T_c$ where $c = (a + b) \bmod M$.

Your task is to find the sum of two given strings A and B over the alphabet of small English letters.

Input

The input file consists of two lines containing strings A and B of the same length. Both strings consist of small letters of English alphabet. The length of each string doesn't exceed 100 000.

Output

Output the sum of strings A and B.

Example

aplusb.in	aplusb.out
`neerc` `neerc`	`aijie`
`z` `y`	`x`

Problem B. Beer Refrigerator

Input file:	`beer.in`
Output file:	`beer.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Beer Lovers Club makes regular parties. They hate warm beer, but club's refrigerator is too small to store enough beer for the whole company. So they decided to order a special super-big beer refrigerator. The new refrigerator should be a parallelepiped $a \times b \times c$ and store exactly n cubical $1 \times 1 \times 1$ beer boxes (the club has n members). To decrease losses of cold, the total area of the surface of the refrigerator must be as small as possible.

For example, if the capacity of the refrigerator must be 12, the possible variants are:

Dimensions	Surface Area
$3 \times 2 \times 2$	32
$4 \times 3 \times 1$	38
$6 \times 2 \times 1$	40
$12 \times 1 \times 1$	50

The best variant in this case is $3 \times 2 \times 2$.

Help the beer lovers to find the optimal dimensions for their new refrigerator.

Input

The input file contains single integer number n ($1 \le n \le 10^6$) — the capacity of the refrigerator.

Output

Output three integer numbers: a, b and c — the optimal dimensions of the refrigerator. If there are several solutions, output any of them.

Example

beer.in	beer.out
`12`	`3 2 2`
`13`	`1 13 1`
`1000000`	`100 100 100`

Problem C. Crosses and Crosses

Input file:	`crosses.in`
Output file:	`crosses.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

The game of *Crosses and Crosses* is played on the field of $1 \times n$ cells. Two players make moves in turn. Each move the player selects any free cell on the field and puts a cross '$\times$' to it. If after the player's move there are three crosses in a row, he wins and the game stops.

You are given a position in the game. Find out, who has won the game.

Input

The input file contains one line containing only characters '`x`' (ASCII code 120) denoting crosses and '`.`' (ASCII code 46) denoting empty cells. The number of cells in the given position does not exceed 30.

Output

The first line of output file should contain one of the following strings:

"`First`", if the first player has won the game;
"`Second`", if the second player has won the game;
"`Nobody`", if nobody has won yet;
"`Invalid`", if the position is invalid, i.e. it could not have been reached in a correctly played game.

Example

crosses.in	crosses.out
`x.xxx.x`	`First`
`xxxxxxxx`	`Invalid`

Problem D. Domestic Networks

Input file:	`domestic.in`
Output file:	`domestic.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Alex is a system administrator of *Domestic Networks Inc.* His network connects apartments and spans over multiple buildings.

The network expands and Alex has to design a new network segment. He has a map that shows apartments to connect and possible links. Each link connects two apartments and for each possible link its length is known. The goal is to make all apartments connected (possibly through other apartments).

Domestic Networks Inc. buys cable in the nearest cable shop. Unfortunately, shop sells only category 5 and 6 cables at price of p_5 and p_6 rubles per meter respectively. Moreover, there are only q_5 meters of category 5 cable and q_6 meters of category 6 cable available in the shop.

Help Alex to solve a hard problem: make a new network construction plan with possible minimal cost. A plan consists of list of links to be made and cable category for each link (each link should be a single piece of cable of either 5 or 6 category). The cost of the plan is the sum of cost of all cables. The total length of cables of each category used in the plan should not exceed the quantity of the cable available in the shop.

Input

The first line of the input file contains two numbers: n — the number of apartments to be connected and m — the number of possible links ($1 \le n \le 1000$, $1 \le m \le 10\,000$).

Following m lines contain possible link descriptions. Each description consists of three integer numbers: a_i — appartments that can be connected by the link and l_i — link length in meters ($0 \le l_i \le 100$). Apartments are numbered from 1 to n.

The last line of the input file contains four integer numbers: p_5, q_5, p_6 and q_6 — price and quantity of category 5 and 6 cables respectively ($1 \le p_i, q_i \le 10\,000$).

Output

If all apartments can be connected with the available cable, output n lines — an optimal network construction plan. The first line of the plan must contain plan's cost. Other lines of the plan must consist of two integer numbers each: a_i and c_i — number of the link to make and c_i — the category of the cable to make it of. Links are numbered from 1 to m in the order they are specified in the input file. If there are more than one optimal plans, output any of them.

If there is no plan meeting all requirements, output a single word "`Impossible`".

Example

domestic.in	domestic.out

6 7
1 2 7
2 6 5
1 4 8
2 3 5
3 4 5
5 6 6
3 5 3
2 11 3 100
	65
	1 5
	2 6
	4 6
	5 6
	7 5

Problem E. Elevator

Input file:	`elevator.in`
Output file:	`elevator.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Edward works as an engineer for *Non-trivial Elevators: Engineering, Research and Construction* (*NEERC*). His new task is to design a brand new elevator for a skyscraper with h floors.

Edward has an idée fixe: he thinks that four buttons are enough to control the movement of the elevator. His last proposal suggests the following four buttons:

- Move a floors up.
- Move b floors up.
- Move c floors up.
- Return to the first floor.

Initially, the elevator is on the first floor. A passenger uses the first three buttons to reach the floor she needs. If a passenger tries to move a, b or c floors up and there is no such floor (she attempts to move higher than the h-th floor), the elevator doesn't move.

To prove his plan worthy, Edward wants to know how many floors are actually accessible from the first floor via his elevator. Help him calculate this number.

Input

The first line of the input file contains one integer h — the height of the skyscraper ($1 \le h \le 500\,000$).

The second line contains three integers a, b and c — the parameters of the buttons ($1 \le a, b, c \le 100\,000$).

Output

Output one integer number — the number of floors that are reachable from the first floor.

Example

elevator.in	elevator.out
15 4 7 9	9

Problem F. Formula

Input file:	`formula.in`
Output file:	`formula.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Nick is a mathematician and his speciality is Boolean logic, especially repetition-free formulas. Formula is repetition-free if each variable occurs in the formula only once.

Let us fix the syntax of considered logical formulae:

- Variables — letters from '`a`' to '`z`' and from '`A`' to '`Z`';
- Parentheses — if E is a formula, then (E) is another;
- Negation — $\neg E$ is a formula for any formula E;
- Conjunction — $E_1 \wedge E_2 \wedge \cdots \wedge E_n$.
- Disjunction — $E_1 \vee E_2 \vee \cdots \vee E_n$.

The operations are listed from the highest priority to the lowest.

An assignment is a function that maps each variable to its Boolean value. Assignment satisfies Boolean formula F if the value of F for this assignment is true.

The problem is to count the number of assignments that satisfy a given repetition-free Boolean formula.

Input

The only line of the input file contains the Boolean formula — a string consisting of characters '`a`'..'`z`', '`A`'..'`Z`', '`(`', '`)`', '`~`', '`&`' and '`|`'. The last three tokens stand for $\neg$, $\wedge$ and $\vee$ respectively. Uppercase and lowercase letters represent different variables. Tokens can be separated by an arbitrary number of spaces. The line contains 1 000 characters at most. The formula in the file is a syntactically correct repetition-free formula.

Output

The first line of the output file must contain the number of variable assignments that satisfy the repetition-free formula given in the input file.

Example

formula.in	formula.out
`A | a & b`	`5`
`~a&~c&~m`	`1`

Problem G. Given a string...

Input file:	`given.in`
Output file:	`given.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Peter's colleague Andrew also works at *RIGS*. Some time ago Andrew invented another definition of sum of two strings of equal length n. In Andrew's definition, the basic alphabet to define this operation consists only of zeros and ones. The sum of two strings $a + b$ is just a string c where $c_i = a_i \vee b_i$ (S_i denotes i-th character of string S). Here $\vee$ stands for usual *OR* operation which returns 1 if and only if at least one of its arguments is 1.

Now Peter must study properties of *closure* of a given string S. The closure of S (denoted $S^{\otimes}$) is a set of strings $S_{(k)} + S_{(l)}$ for any $0 \le k, l \le n - 1$, where n is the length of S, and $S_{(k)}$ denotes an operation of *k-th circular shift* of S — moving k last characters from the end of the string S to its beginning. For example, the second circular shift of `abcde` is `deabc`.

Given a string T, Peter's task is to check whether it belongs to $S^{\otimes}$. Could you solve this task for him?

Input

The first line of the input file contains a given string T. The second line contains S. Both strings are of equal length in range from 1 to 100 characters. All characters in these strings are zeros or ones.

Output

If a given string belongs to $S^{\otimes}$, output "`Yes`". Otherwise output "`No`".

Example

given.in	given.out
11110 10101	Yes
101111 101010	No

Problem H. History of Football

Input file:	`history.in`
Output file:	`history.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Henry is a historian. He specializes in the history of sports, especially football. Whenever he sees a table of a football tournament, he saves it into his database.

Recently he ran across a web-site with standings of a small tournament. Unfortunately for him, the results of the games were lost, and the only available information was the amount of points gained by each team.

Disappointed by that, he decides to have some mathematical fun and to calculate in how many different ways the games of the championship could have ended. He doesn't care about the scores of the games, he only cares about the winners.

In that tournament the following rules were applied:

- Each team plays against each other team exactly once.
- In case of a tie each team gains 1 point.
- In other case the winner gains 3 points and the loser gains 0 points.

For example, if Henry knows that each of 3 teams had got 3 points by the end of the tournament, the answer to his question is that there are two possible tournament tables:

Possible table number 1

Team	A	B	C	Points
A	-	3	0	3
B	0	-	3	3
C	3	0	-	3

Possible table number 2

Team	A	B	C	Points
A	-	0	3	3
B	3	-	0	3
C	0	3	-	3

Help Henry calculate the number of different possible tournament tables (without consideration of the scores of the games).

Input

Input file contains integer n, the number of teams in the championship ($2 \le n \le 5$). The following n lines contain one integer number each — points gained by the teams.

Output

Output one integer number — the number of possible tournament tables with given total points. It is guaranteed that there is at least one such tournament table.

Example

history.in	history.out
3 3 3 3	2

Problem I. iChess

Input file:	`ichess.in`
Output file:	`ichess.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

The Jury of *NEERC'07* quarterfinals is proud to present you a new game — chess patience. This patience is played not with cards, but with black and white square tiles. The goal of the game is to place these tiles on a flat surface so that they form a square colored in a chess-like pattern. The square should be totally filled and be of the maximal possible size. There may remain some spare tiles, if they do not fit into the resulting square.

To make this game more popular, a computer version of this patience named *iChess* was developed. The rules are the same with the exception that the player is given the number of tiles, not the actual tiles. Also, the result of the patience is not the actual layout, but the side length (measured in tiles) of the maximal square with the required layout.

Your task is to write a program which can play *iChess* patience.

Input

The input file contains two integer numbers b and w — the number of black and white tiles respectively ($0 \le b, w \le 10\,000$).

Output

The first line of the input file must contain a single integer number s — the side length of the maximum possible square made of at most b black and w white tiles.

If no square can be formed with the given tiles, output a single word "`Impossible`".

Example

ichess.in	ichess.out
12 15	5
0 0	Impossible

Problem J. Journey with Pigs

Input file:	`journey.in`
Output file:	`journey.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Farmer John has a pig farm near town A. He wants to visit his friend living in town B. During this journey he will visit n small villages so he decided to earn some money. He tooks n pigs and plans to sell one pig in each village he visits.

Pork prices in villages are different, in the j-th village the people would buy a pork at p_j rubles per kilogram. The distance from town A to the j-th village along the road to town B is d_j kilometers.

Pigs have different weights. Transporting one kilogram of pork per one kilometer of the road needs t rubles for addition fuel.

Help John decide, which pig to sell in each town in order to earn as much money as possible.

Input

The first line of the input file contains integer numbers n ($1 \le n \le 1000$) and t ($1 \le t \le 10^9$). The second line contains n integer numbers w_i ($1 \le w_i \le 10^9$) — the weights of the pigs. The third line contains n integer numbers d_j ($1 \le d_j \le 10^9$) — the distances to the villages from the town A. The fourth line contains n integer numbers p_j ($1 \le p_j \le 10^9$) — the prices of pork in the villages.

Output

Output n numbers, the j-th number is the number of pig to sell in the j-th village. The pigs are numbered from 1 in the order they are listed in the input file.

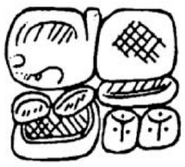
Example

journey.in	journey.out

3 1
10 20 15
10 20 30
50 70 60
	3 2 1

Problem K. K’ak’-u-pakal and Mayan Script

Input file:	`kak-u-pakal.in`
Output file:	`kak-u-pakal.out`
Time limit:	2 seconds
Memory limit:	64 megabytes



When scientists re-discovered ancient Mayan cities, they found many texts, written in unknown script. An example of the script is shown on the right, where the name of *K’ak’-u-pakal*, a military leader and priest in ancient Mayan city of *Chichén Itzá*, is written (see A. W. Voss, H. J. Kremer, *K’ak’-u-pakal, Hun-pik-tok’ and the Kokom* for details). This hieroglyph can be found in many sites of the city.

Mayan hieroglyphs are not hieroglyphs in proper sense, but compositions of separate glyphs. All known glyphs (there is about one thousand of them) are indexed with numbers from 1 to 9999. A special language for encoding glyphs relative positions made it possible to write any hieroglyph in plain text. For example the above *K’ak’-u-pakal* hieroglyph is encoded as “`((669:604).(586:(27:(534.534))))`”.

Here is the formal grammar of the language (adapted for the contest):

$$\begin{array}{rcl} \langle\text{inscription}\rangle & \longrightarrow & \langle\text{glyph id}\rangle \mid \text{‘}\texttt{(}\text{’}\,\langle\text{inscription}\rangle\,\text{‘}\texttt{.}\text{’}\,\langle\text{horisontal group}\rangle\,\text{‘}\texttt{)}\text{’} \mid \\ & & \quad \text{‘}\texttt{(}\text{’}\,\langle\text{inscription}\rangle\,\text{‘}\texttt{:}\text{’}\,\langle\text{vertical group}\rangle\,\text{‘}\texttt{)}\text{’} \\ \langle\text{horisontal group}\rangle & \longrightarrow & \langle\text{inscription}\rangle\ [\text{‘}\texttt{.}\text{’}\,\langle\text{horisontal group}\rangle] \\ \langle\text{vertical group}\rangle & \longrightarrow & \langle\text{inscription}\rangle\ [\text{‘}\texttt{:}\text{’}\,\langle\text{vertitcal group}\rangle] \end{array}$$

The hieroglyph encoding describes a process of the hieroglyph composition. Glyphs are combined horizontally or vertically (using ‘`.`’ or ‘`:`’) into blocks, which in turn are combined into larger and larger blocks, until the necessary configuration is achieved.

It took a hundred years to decipher Mayan hieroglyphs and convert them into plain text, but we hope that you can create a program for the backward conversion (text to hieroglyph layout) much quicker.

Input

The first line of the input file contains a space-free text string (255 symbols at most), containing Mayan hieroglyph encoded in plain text form.

Output

Output text, composed of characters ‘`+`’, ‘`-`’, ‘`|`’, ‘ ’ (ASCII codes 43, 45, 124, 32), ‘`0`’..‘`9`’ and line feeds. All blocks of a group should have exactly the same size. The glyph id (with one leading and one trailing space) should be placed at the upper left corner of the block. The output should be as short as possible. It’s guaranteed that for each test there exists a proper layout, which has 100 000 bytes at most.

Example

`kak-u-pakal.in`

```
((669:604).(586:(27:(534.534))))
```

`kak-u-pakal.out`

```
+-----------+-----------+
| 669       | 586       |
|           |           |
|           |           |
+-----------+-----------+
| 604       | 27        |
|           +-----+-----+
|           | 534 | 534 |
+-----------+-----+-----+
```