ACM ICPC SOUTH PACIFIC DIVISIONALS

OCTOBER 8, 2016

Contest Problems

A: Analysis of Advanced Analytics
B: Balloon Connect-the-Dots
C: Candy Sales
D: Dangerous Discus
E: Exciting Startup
F: Fusion
G: Grass Cutting
H: Haggling With a Witcher
I : Intuidiff
J : Jousting Tournament
K: Killer Sudoku
L: Loading Cargo

This contest contains twelve problems. Twelve teams will advance to the Regional Finals.

For problems that state "*Your answer should have an absolute or relative error of less than* 10^{-6}", your answer, a, will be compared to the correct answer, b. If $|a - b| < 10^{-6}$ or $\frac{|a-b|}{|b|} < 10^{-6}$, then your answer will be considered correct.

Judging team:
Darcy Best, Mike Cameron-Jones, Malcolm Corney, Tim French, Walter Guttmann, Andrew Haigh, Henning Koehler, Richard Lobb, Kourosh Neshatian, Evgeni Sergeev, Kevin Tran, Max Ward-Graham

Problem A
Analysis of Advanced Analytics
Time limit: 1 second


Source: Pexels

James is purchasing lecture notebooks for his upcoming course: *Analysis of Advanced Analytics*. James knows exactly how many pages are needed for each day during the semester. On the first day, he starts writing his notes at the beginning of the first notebook. On the second day, he checks if there are enough pages left in the first notebook to write all of the notes for the second day. If there are enough pages left, he will write the notes in the first notebook. If there are not enough pages left, he will start a new notebook and write all of his notes for that lecture in the new notebook. He continues this process, checking every day if there is enough room left in the current notebook to write the notes for that day. If there is enough room, he will continue writing in that notebook and if there is not enough room, he will start a new notebook.

Each notebook contains the same number of pages. Given the lengths of the notes for each day during the entire semester, how many notebooks should James buy to contain all of the notes for the semester?

Input

The input consists of two lines. The first line contains two integers n ($1 \leq n \leq 1\,000$), which is the number of days in the semester, and k ($1 \leq k \leq 1\,000$), which is the number of pages in each notebook.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq k$), where a_i is the number of pages needed for the i^{th} day in the semester.

Output

Display the number of notebooks that James needs for the entire semester.

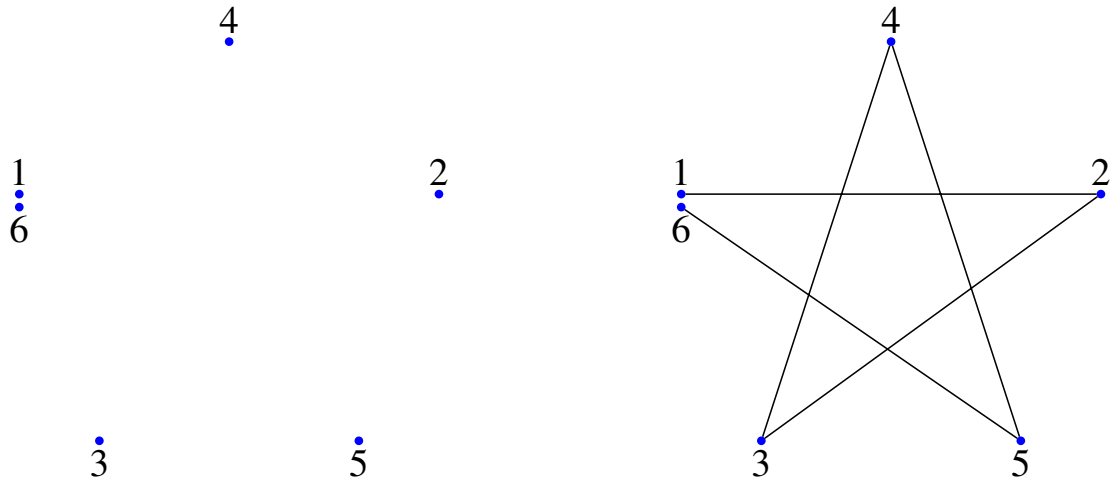
Sample Input 1	Sample Output 1
4 100 60 30 20 60	2

Sample Input 2	Sample Output 2
3 100 30 100 10	3

This page is intentionally left (almost) blank.

Problem B
Balloon Connect-the-Dots
Time limit: 2 seconds

Have you ever played around with *connect-the-dots*? For those of us who are not artistic, this is one of the only ways to draw amazing pictures. For a connect-the-dots, there are n dots on the page, each labelled with an integer between 1 and n. You are to draw $n-1$ straight lines: one between 1 and 2, one between 2 and 3, one between 3 and 4, and so on. In the end, the image you have should be beautiful. Here is an example before and after you connect the dots:



Today, you are going to (hopefully) help me draw a connect-the-dots on the surface of a balloon. For simplicity, we are going to assume that the balloon is a perfect sphere. To begin with, the balloon has a radius of 1 centimetre. I have drawn n dots on the surface of the balloon and labelled each of them from 1 to n. As I inflate the balloon, the points will start moving further and further apart. My goal is to end up with as large a picture as possible (that is, the total length of the $n-1$ line segments should be as large as possible).

Because I do not want the image to look faded, I will first inflate the balloon and then (and only then) draw the $n-1$ lines. I can increase the volume of the balloon by v cm^3 per minute and I can draw at a speed of d cm per minute. When I draw the line from point i to $i+1$, I will draw a straight line along the surface of the balloon. Each line will be the shortest line between the two points (in certain cases, there may be multiple shortest lines—which one is drawn is not important for this problem since we are only concerned with the lengths of the lines).

I have T total minutes to spare, so the time to both inflate the balloon and draw the lines must be at most T minutes. When I finish drawing, all of the $n-1$ lines must be fully drawn (completed). What is the largest sum of line lengths that I can draw?

Input

The input starts with a line containing four integers n, T, v and d, where n ($2 \leq n \leq 1\,000$) is the number of points, T ($1 \leq T \leq 1\,000\,000$) is the time available to both inflate the balloon and draw the $n-1$ lines, v ($1 \leq v \leq 1\,000$) is the speed in which I can inflate the balloon (in cm^3 per minute) and d ($1 \leq d \leq 1\,000$) is the speed in which I can draw a line (in cm per minute).

The next n lines describe the points on the balloon in order from 1 to n, assuming that the balloon's centre is at the origin. Each of these lines consists of two real numbers x and y and an integer s ($s \in \{-1, 1\}$), giving the point's position on the surface of the ballon at (x, y, z), where $z = s\sqrt{1-x^2-y^2}$. It is guaranteed that $0 \leq x^2+y^2 \leq 1$ and x and y are in centimetres.

Note that lines may intersect one another (or may overlap completely), and so you may count a certain line segment multiple times (see Sample Input 1 for example). Each x and y will be given to no more than 6 digits of precision. It is guaranteed that I can draw the image on the balloon of radius one in at most T minutes.

Output

Display the length (in centimetres) of the largest completed drawing on the balloon that I can make. Your answer should have an absolute or relative error of less than 10^{-6}.

Notes

Be careful when $x^2 + y^2 = 1$. When trying to evaluate $\sqrt{1 - x^2 - y^2}$, the value of $1 - x^2 - y^2$ may be stored as $-0.000 \cdots 0001$, so $\sqrt{1 - x^2 - y^2}$ may fail (you cannot take the square root of a negative number).

Sample Input 1

```
3 10 10 20
1 0 1
0 1 1
1 0 1
```

Sample Output 1

```
9.03600166336021
```

Sample Input 2

```
2 10 1 1
0.3 0.5 1
0.2 0.4 -1
```

Sample Output 2

```
2.86454577269227
```

Sample Input 3

```
3 10 1 1
0.3 0.5 1
0.3 0.5 1
0.2 0.4 -1
```

Sample Output 3

```
2.86454577269227
```

Problem C
Candy Sales
Time limit: 6 seconds

Cathy's favourite candy brand is releasing n new flavours of candy: one new flavour on each of the next n days. A pack of candy flavour i (the new flavour released on the i^{th} day) will cost w_i dollars on the day that it is released, but in order to encourage customers to try the new flavours, each candy flavour increases in price by one dollar on each day after it has been released. Precisely, on day $j \geq i$, a pack of the candy flavour released on day i costs

$$w_i + (j - i)$$

dollars.


Source: By Tiia Monto, CC BY-SA 3.0

Cathy wants to buy exactly one pack of candy on each one of the next n days and wants to get the most candy for her money. Calculate the price of the cheapest pack of candy available on each of the next n days. There is an unlimited number of packs of each flavour of candy and each flavour is available on the day of its release and all subsequent days.

Input

The input consists of two lines. The first line contains an integer n ($1 \leq n \leq 200\,000$), which is the number of flavours. The second line contains n integers $w_1, w_2, ..., w_n$ ($1 \leq w_i \leq 100\,000$), where w_i is the initial price of the i^{th} candy flavour in dollars.

Output

Display n integers, the i^{th} of which is the price of the cheapest pack of candy that Cathy can buy on day i.

Sample Input 1

```
4
3 6 7 4
```

Sample Output 1

```
3 4 5 4
```

Sample Input 2

```
4
22 22 28 24
```

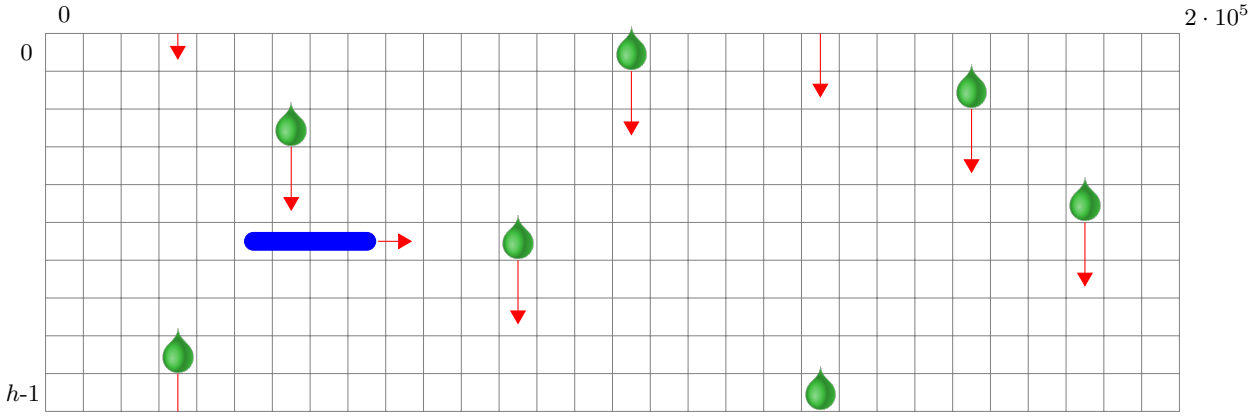
Sample Output 2

```
22 22 23 24
```

This page is intentionally left (almost) blank.

Problem D
Dangerous Discus
Time limit: 4 seconds

You are playing a computer game in which you need to throw discs horizontally from the left side of the screen to the right. However, acid is constantly dripping vertically from multiple points in the ceiling to the ground, and any contact with the deadly acid will instantly destroy your disc. After several fails, you wonder whether it is possible to make it through to the other side.



There are n dripping spots located on row 0. Any drop that originates from dripping spot k falls v pixels every time your disc moves one pixel to the right. When a drop hits the ground, a new drop starts to fall from the dripping spot in the same column so that there is always exactly one acid drop per column. The drop present in the column of dripping spot k at time t has a vertical position of

$$(y_k + t \cdot v) \mod h,$$

where y_k is the vertical position of the k^{th} drop at time 0 and h is the height of the ceiling.

Animation of your disc and the acid drops alternate. That is, first your disc moves one pixel horizontally to the right, then all acid drops move v pixels vertically downwards, visiting each pixel on the way. Your disc is only a single pixel thick and w pixels wide, while acid drops occupy a single pixel. The disc is destroyed if it ever occupies the same pixel as an acid drop. The throwing height can be any height between 0 and $h - 1$, inclusive, and the disc will stay at the same height throughout its journey. At time 0, the disc is occupying columns $-w$ through -1, inclusive. The disc's journey is complete once its left side has reached column 200 000.

Input

The input starts with a line containing four integers n, h, v and w, where n ($1 \leq n \leq 200\,000$) is the number of dripping points, h ($1 \leq h \leq 200\,000$) is the ceiling height, v ($1 \leq v \leq 500$) is the dripping speed and w ($1 \leq w \leq 500$) is the width of the disc.

The next n lines describe the acid drops. Each of these lines contain two integers x ($0 \leq x < 200\,000$), which is the column of the acid drop, and y ($0 \leq y < h$), which is the initial vertical position of the acid drop.

No two acid drops will have the same x value.

Output

If it is possible to throw the disc through the acid rain without getting destroyed, display `VICTORY`. Otherwise, display `GAME OVER`.

Sample Input 1

```
2 10 2 3
3 4
5 8
```

Sample Output 1

```
VICTORY
```

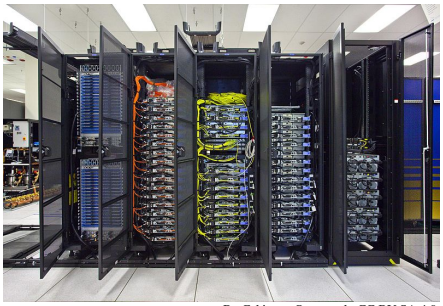
Sample Input 2

```
1 5 2 2
1 1
```

Sample Output 2

```
GAME OVER
```

Problem E
Exciting Startup
Time limit: 3 seconds



By Cskiran - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=46569840

Alice is founding her own exciting startup, a very trendy messaging app where users can send each other messages that may be up to (and including) n characters long. Now she just needs some users!

Cathy is Alice's first user. She wants to send a string containing $t > n$ characters by breaking it up into smaller messages each containing no more than n characters. However, to ensure the sequence of messages can be understood correctly, she wants each message to indicate its position in the message by appending to any content an indicator of the exact form `'_a/b'` indicating this is the a^{th} message out of b total. Obviously, this indicator also counts towards the limit of n characters in each message.

Suppose $n = 7$. If Cathy wishes to send the string "floccinaucinihilipilification" ($t = 29$), she needs to send at least 20 messages. One way to do this is to send the messages `fl_1/20`, `oc_2/20`, `ci_3/20`, `na_4/20`, `uc_5/20`, `in_6/20`, `ih_7/20`, `il_8/20`, `ip_9/20`, `i_10/20`, `l_11/20`, `i_12/20`, `f_13/20`, `i_14/20`, `c_15/20`, `a_16/20`, `t_17/20`, `i_18/20`, `o_19/20`, `n_20/20`.

Cathy wants you to calculate for a given n and t if it is possible to send her chosen string, and if so, the minimum number of messages in total that she has to send.

Input

The input consists of a single line containing two integers n ($5 \leq n \leq 100$), which is the maximum length of each message on Alice's messaging app, and t ($n < t \leq 10^6$), which is the length of the string that Cathy wishes to send.

Output

Display the minimum number of messages that Cathy requires to send her string. If the string cannot be sent with any number of messages, display `-1` instead.

Sample Input 1	Sample Output 1
`7 26`	`9`

Sample Input 2	Sample Output 2
`7 29`	`20`

Sample Input 3	Sample Output 3
`5 1000000`	`-1`

Sample Input 4	Sample Output 4
`12 5000`	`1947`

Sample Input 5	Sample Output 5
`12 5001`	`1947`

This page is intentionally left (almost) blank.

Problem F
Fusion
Time limit: 15 seconds

What a time to be alive! Today, the very first *cold fusion* reactor has just been built and cleared for operation. Of course, when the reactor is started for the first time, nothing happens.

Bewildered physicists pore over their calculations looking for errors. Countless double-checks and triple-checks are made before a trio of engineers discover the problem: two atoms in the reactor have become α-stuck!

The reactor can be thought of as a grid of cells with R rows and C columns. Each cell is either empty, contains a deuterium atom (two of which are α-stuck) or is blocked off by a control rod to prevent runaway reactions. Each cell is adjacent to the four cells to its left, right, top and bottom (cells on the edge of the reactor are adjacent to fewer than four cells).

You can issue fusion instructions to the reactor by specifying two atoms to be fused. Be wary; two atoms can be fused only if they are in adjacent cells or there is a path of adjacent empty cells from one atom to the other. When two atoms are fused, they will produce helium which floats away, effectively removing the two original atoms from the reactor (leaving the cells empty).

In order to fix the problem in the reactor, you will need to fuse the α-stuck atoms with each other. What is the fewest number of instructions you need to give to the reactor in order to fuse the α-stuck atoms?

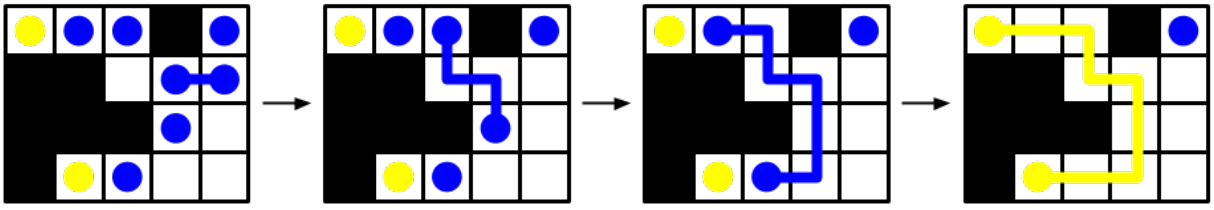


Figure F.1: This is the reactor for Sample Input 1. Black squares denote blocked off cells, blue circles denote deuterium atoms and the yellow circles denote the two α-stuck atoms. One optimal sequence of fusions is shown.

Will you and your fellow engineers overcome the final barrier to *cold fusion*?

Input

The first line of input contains two integers R ($1 \leq R \leq 1000$), which is the number of rows in the reactor, and C ($1 \leq C \leq 1000$), which is the number of columns in the reactor. The next R lines describe the reactor. Each of these lines is a string with exactly C characters.

- `'.'` denotes an empty cell.
- `'#'` denotes a blocked cell.
- `'O'` denotes a cell containing a deuterium atom.
- `'A'` denotes a cell containing an α-stuck atom.

There are exactly two `'A'`s in the input.

Output

Display the minimum number of fusion instructions needed to fuse the α-stuck atoms. If it is not possible to fuse the α-stuck atoms together, display -1 instead.

Sample Input 1	Sample Output 1
4 5 AOO#O ##.OO ###O. #AO..	4

Sample Input 2	Sample Output 2
2 5 ..O.. A###A	-1

Problem G
Grass Cutting
Time limit: 1 second


Source: Pixabay

As a computer scientist, one of my least favourite tasks is mowing the lawn. Every week, you have to drag out the machine and go back and forth in your backyard until every blade of grass is cut. A few months back, I got tired of doing this and started only cutting portions of my lawn. My lawn can be thought of as a 10×10 grid, forming 100 sections of grass. Each week, I selected 3 rows and 3 columns and cut the grass in all of the sections in them. Over time, I figured that the height of the lawn would be roughly equal.

When I started this new technique, every section of grass was 1 cm high. Each week, every section grows by 1 cm and then I cut some sections of grass as described above. When I cut a section of grass, its height becomes 1 cm. Given which rows and columns I have cut over the past few weeks, can you tell me the height of each section of lawn?

Input

The first line of the input contains an integer k ($1 \leq k \leq 100$), which is the number of weeks that I have performed this new cutting technique.

The next k lines describe the weeks. Each of these lines contain six integers. The first three integers on each line are the three rows that I selected for that week and the next three integers are the three columns that I selected for that week. Each of these numbers is between 1 and 10, inclusive. The three rows will be distinct and the three columns will be distinct.

Output

Display a 10×10 grid of numbers indicating the height of the grass immediately after the last cutting. The top row is row 1 and the bottom row is row 10. The left column is column 1 and the right column is column 10.

Sample Input 1

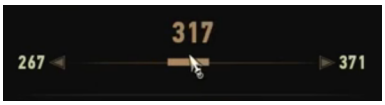
```
1
1 2 3 4 5 6
```

Sample Output 1

```
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
2 2 2 1 1 1 2 2 2 2
2 2 2 1 1 1 2 2 2 2
2 2 2 1 1 1 2 2 2 2
2 2 2 1 1 1 2 2 2 2
2 2 2 1 1 1 2 2 2 2
2 2 2 1 1 1 2 2 2 2
2 2 2 1 1 1 2 2 2 2
```

Sample Input 2	Sample Output 2
2 1 2 3 4 5 6 1 3 5 4 8 10	1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 2 3 1 3 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 2 3 1 3 1 3 3 3 1 2 2 3 1 3 1 3 3 3 1 2 2 3 1 3 1 3 3 3 1 2 2 3 1 3 1 3 3 3 1 2 2 3 1 3 1

Problem H
Haggling With a Witcher
Time limit: 2 seconds



You are playing a video game called "The Association for Control of Monsters" (ACM). In this game, you work for the ACM as a *witcher*, and your name is Jerry. As a witcher, it is your job to kill hideous, and fictional, monsters with your silver sword. But you do not do this for free, instead you demand a fee for your work. When a non-playing character in the game (an NPC) has a contract for you, you will attempt to haggle with them.

Haggling is carried out as follows. The NPC will choose an integer amount of gold pieces, t, between L and R, inclusive. The NPC chooses t uniformly at random before the haggling starts. You will offer them a fee and they will accept the offer you make if it is at most t, otherwise they will reject the offer, after which you may make another offer. Each offer must be an integer value between L and R, inclusive. Watching the dialogue with the NPC when your offer is rejected or accepted takes 100 milliseconds. Since you are a hardcore gamer having a very short attention span, you get bored after T milliseconds.

You have saved the game state just as the haggling started, which means that if your offer is accepted by the NPC, you may reload this save and start again with the knowledge you gained (note that the value of t does not change when you reload the game). However, reloading the save also takes 100 milliseconds on top of the 100 milliseconds for the NPC's dialogue. Note that if your offer is rejected, you can make another offer, and if that second offer is rejected, you can make another, and so on until you run out of time. If your offer is accepted, however, you may choose to either take the gold or reload the save. If you ever get bored (that is, use more than T milliseconds), then you must stop what you are doing (you might be half way through watching accept or reject dialogue, or loading a save) and you get no gold. To be the very best hardcore gamer, you must be able to haggle well, so you have decided to write a program to help. Given L, R and T, what is the highest expected value of gold you can get?

The expected value is the average amount of gold over an infinite number of different rounds of haggling. For example, say that $L = 2$, $R = 4$, $T = 250$ and our strategy is to make an offer of 4, then if it is rejected, make an offer of 2. This has an expected value of $\frac{8}{3}$ gold. This is because the threshold value t picked by the NPC is equally likely to be $2, 3$ or 4 over an infinite number of runs. This strategy earns 2 when $t = 2, 3$ and earns 4 when $t = 4$, so it has expected value $\frac{2+2+4}{3} = \frac{8}{3}$.

Input

The input consists of a single line containing three integers L ($1 \leq L \leq 80$), R ($L \leq R \leq 80$) and T ($1 \leq T \leq 10\,000$) as described above.

Output

Display the highest expected value you can attain. Your answer should have an absolute or relative error of less than 10^{-6}.

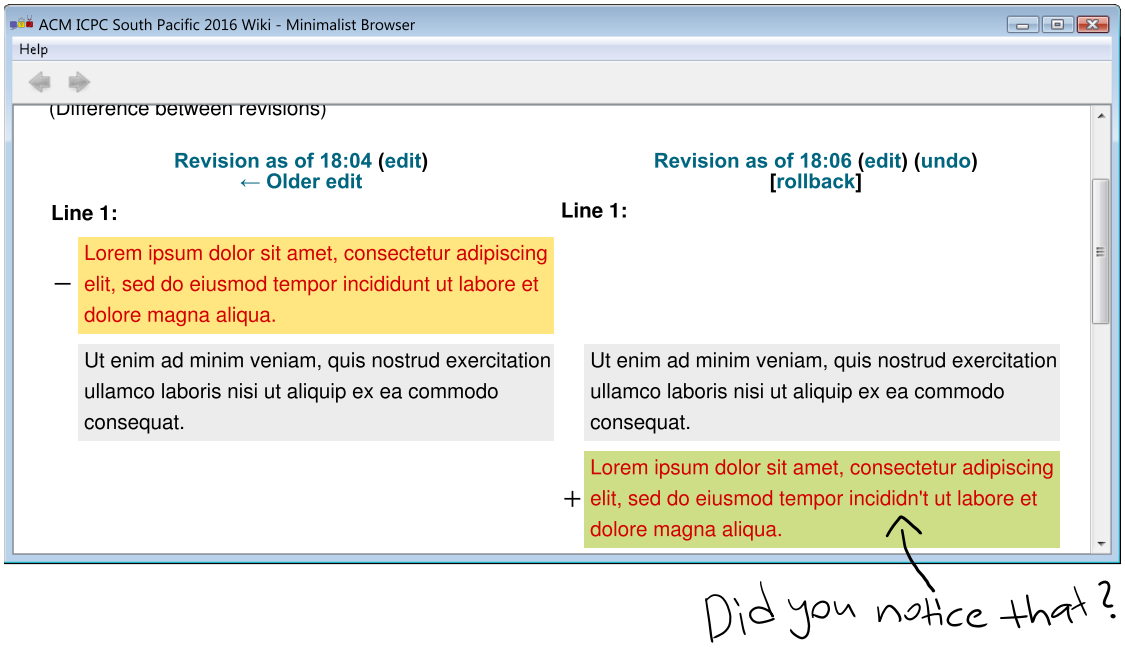
Sample Input 1	Sample Output 1
2 4 250	2.6666666666666667

Sample Input 2	Sample Output 2
1 5 100	1.8

This page is intentionally left (almost) blank.

Problem I
Intuidiff
Time limit: 7 seconds

Nowadays, web applications let us compare revision history for posts and articles using some variant of the quintessential `diff` utility that finds a small set of insertions and deletions to transform one document into another.



Here is a typical problem with it. Suppose the original document is made up of two paragraphs, `P1 P2`, and we edit it to become `P2 P1`. Then `diff` might describe the changes as: delete `P1`, leave `P2`, insert some new text. The fact that the new text is equal to `P1` is not used. So, in some sense, `diff` is "linear" and does not work in the most convenient way possible when we re-order parts of the document.

Can we do better? (More importantly, can it be finished today?) We will adopt this basic requirement: when a part of the new version of a document has appeared somewhere in the old version, it should be highlighted to reflect this fact.

To capture this idea, define the *Intuidiff distance* from one document to another as the smallest integer `N`, such that the second document is the concatenation of `N` *blocks*, where each *block* is either a substring of the original document or a single new character. (Note that we do not need to worry about deletions because, unlike `diff`, we do not implicitly begin with the original document—instead, it starts off empty. Thus, the Intuidiff distance from a document to itself is 1. But this is the only counter-intuitive thing about it.)

Given two strings, compute and print the Intuidiff distance from the first to the second.

Input

The input consists of two lines. Each line contains a string whose length is between 1 and 500 000 inclusive. Each string only contains alphanumeric and underscore characters.

Output

Display the Intuidiff distance from the first string in the input to the second string in the input.

Notes

An optimal Intuidiff concatenation for Sample Input 1 is:

```
p + aragraph_ + e + d + i + t + 3 + _ + Paragraph
```

Sample Input 1	Sample Output 1
Paragraph_paragraph paragraph_edit3_Paragraph	9

Problem J
Jousting Tournament
Time limit: 2 seconds


Source: Wikimedia Commons

Jousting is a sport that consists of two knights on horseback attempting to knock the opponent off their horse. Jousting tournaments are becoming more and more popular. Back in the old days, there were only a few jousters, so coming up with a schedule for the tournaments was easy. But now that there are so many participants, a more automated system is required. This is where you come in: you must give a schedule that will determine who is the best in the entire tournament.

There are n competitors in the tournament numbered $1, 2, \ldots, n$. Each jouster will have their own skill level that will remain constant throughout the entire tournament. The winner of each match is the jouster with the higher skill level. For simplicity, we will assume that each jouster has a different skill level, so there are no ties.

The schedule must be given as an ordered sequence of triples. The first two components of each triple should contain the two competitors and the last component of the triple is the winner. The winner must be a *placeholder* (represented by a lowercase letter) and the competitors can be either a number or a placeholder. For example, consider the following match:

```
1 2 a
```

This means that jouster `1` competes against jouster `2` and then `a` will be the number of the winner of that match. From that point onwards, `a` can be used as a competitor. For example, consider the following schedule for a tournament with 4 jousters and 3 matches:

```
1 2 a
3 4 b
a b c
```

The winner of the tournament above is the jouster that is represented by `c`. A placeholder may only be used as a competitor once it has been used as a winner in a previous match. A placeholder cannot be used as both a competitor and the winner in the same match. You may reuse placeholders, but once you set a placeholder to the winner of some joust, that placeholder loses the information from previous matches with that same placeholder as the winner.

Note that a jouster may compete in any number of matches. The figure below gives an illustration for the tournaments given in the sample outputs.

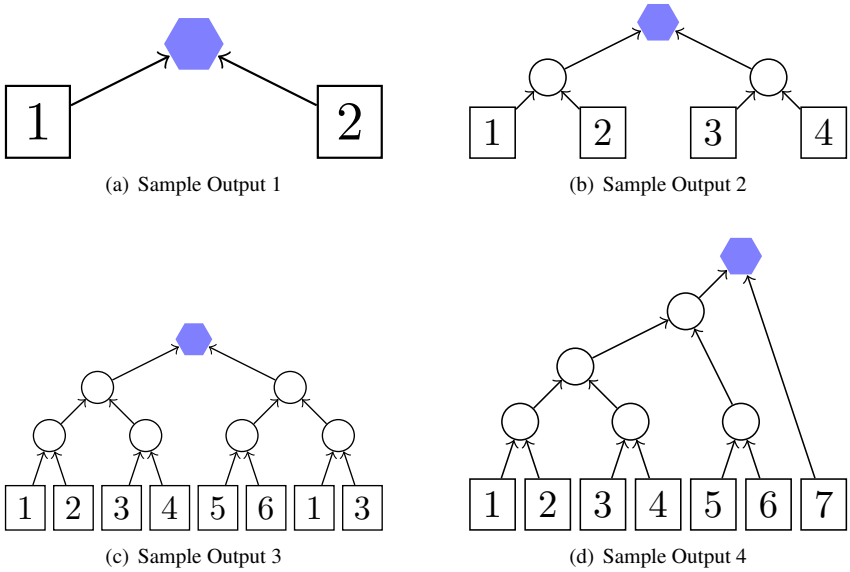
Input

The input consists of a single line containing one integer n ($2 \leq n \leq 1\,000$), which is the number of competitors in the tournament.

Output

Display any valid schedule that will determine the winner of the n jouster tournament. The tournament must consist of no more than $10\,000$ matches. Each match must be represented as 3 space separated items on one line. The only items that can appear are integers $1, 2, \ldots, n$ and lowercase letters `a`, ..., `z`. At the end of the tournament, the winner must be stored in the placeholder `a`. This means that the example above with 4 jousters and 3 matches is not a valid answer since the winner was stored in the placeholder `c`.

It is fine if both competitors in some match are the same. For example, in Sample Output 3, if competitor 1 is the tournament winner, then the final match will be competitor 1 against competitor 1. Any valid schedule will be considered correct.



(a) Sample Output 1

(b) Sample Output 2

(c) Sample Output 3

(d) Sample Output 4

Figure J.1: Sample Outputs

Sample Input 1

```
2
```

Sample Output 1

```
1 2 a
```

Sample Input 2

```
4
```

Sample Output 2

```
1 2 b
3 4 x
x b a
```

Sample Input 3

```
6
```

Sample Output 3

```
1 2 a
3 4 b
5 6 c
1 3 d
a b x
c d y
x y a
```

Sample Input 4

```
7
```

Sample Output 4

```
1 2 a
3 4 b
5 6 c
a b d
c d b
b 7 a
```

Problem K
Killer Sudoku
Time limit: 1 second

Standard Sudoku puzzles have been popular for long enough now that you are probably very familiar with them. However, just in case you have missed them, we will start with a quick recap.

A Sudoku puzzle is a 9×9 grid of squares that must be filled in with the numbers 1 to 9, one number per square, such that each of the nine rows, nine columns and nine *major* sub-grids contain a permutation of the numbers 1 to 9. (The major sub-grids are the nine 3×3 sub-grids that make up the overall 9×9 grid.) A standard Sudoku puzzle starts with some numbers already placed in their squares and the aim is to fill in the rest of the puzzle so as to meet the row, column and major sub-grid constraints. Figure K.1 is an example of a solved standard Sudoku.

2	9	5	7	4	3	8	6	1
4	3	1	8	6	5	9	2	7
8	7	6	1	9	2	5	4	3
3	8	7	4	5	9	2	1	6
6	1	2	3	8	7	4	9	5
5	4	9	2	1	6	7	3	8
7	6	3	5	2	4	1	8	9
9	2	8	6	7	1	3	5	4
1	5	4	9	3	8	6	7	2

Figure K.1: A standard Sudoku.

Killer Sudoku is a popular variant on the standard form, such that instead of starting with some individual numbers already placed in their squares, the grid is divided into one or more connected regions popularly known as *cages* and there are constraints on the numbers that may appear in the cages. Each cage has a total specifying what the number(s) in the cage must sum to. In the sub-variant that we consider here, the number(s) in a cage must also be a set (that is, there must be no repeated numbers in a cage).

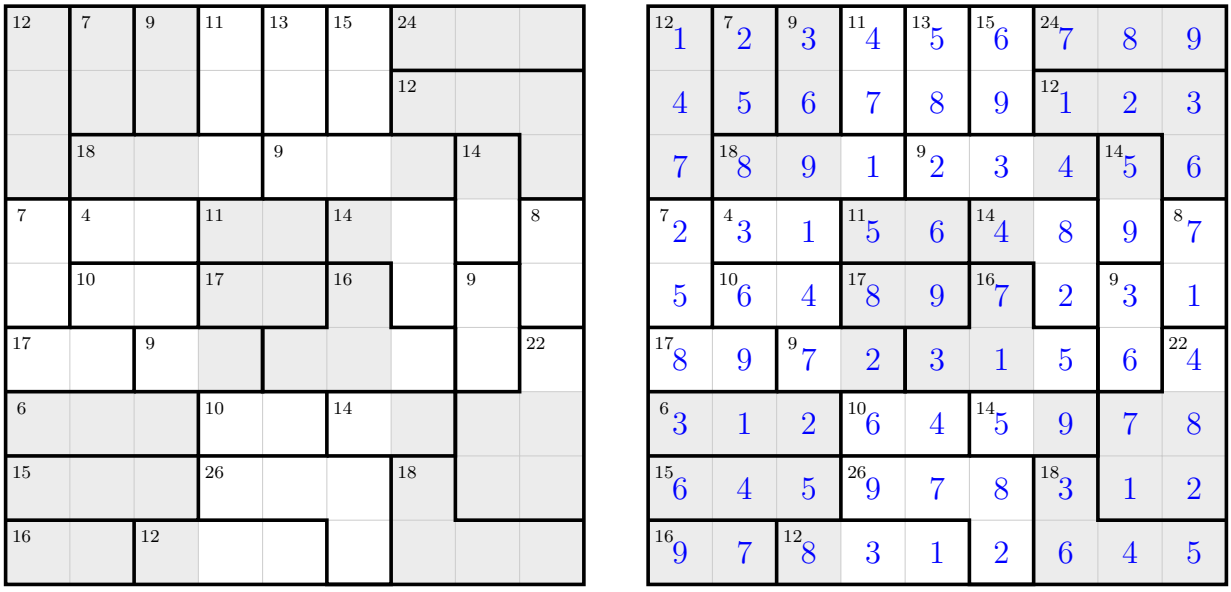


Figure K.2: A Killer Sudoku grid (left) with a solution (right).

Your task is to check whether a completed grid meets all Killer Sudoku constraints for the sub-variant we consider: the row, column, major sub-grid and cage constraints. Note that there is no guarantee that the cages in a grid are such that the grid is soluble as a Killer Sudoku puzzle.

Input

The input starts with a diagram of the grid. The diagram is a 19×37 array of characters.

The grid is a 9×9 grid of (not quite square) "squares", as would be expected. A square, including its border, consists of 3 (vertically aligned) rows of 5 characters each. The border of each square is shared with the relevant adjacent squares, where they exist, and the border is filled in around the grid's edge, as though the exterior were boundaries of an extra cage. The interior of each square consists of a row of 3 characters: a space, the integer N $(1 \leq N \leq 9)$ in the square and another space.

The 4 border corners of a square are each defined by a plus (+) character. Each border side between vertically adjacent squares is defined by 3 characters between the corresponding corners. These 3 characters are all spaces if the two squares sharing the border are in the same cage and otherwise they are all hyphens (-). Each border side between horizontally adjacent squares is defined by a single character (under one corner and above another). This character is a space if the squares sharing the border are in the same cage and is a vertical bar (|) otherwise.

```
+---+---+---+---+---+---+---+---+---+
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+   +   +   +   +   +   +   +   +   +
| 0   0   0   0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+
```

```
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+
```

Figure K.3: The general structure for the grid is given here. Each `0` will be an integer from 1 to 9. The left is the minimal case with one large cage and the right is the maximal case with 81 small cages.

This is followed by one line for each cage in the diagram. Each line contains three integers R $(1 \leq R \leq 9)$, C $(1 \leq C \leq 9)$ and S $(1 \leq S \leq 45)$. R and C specify the row and column of a square within this cage and S specifies the value that the number(s) within this cage must sum to.

The rows are numbered from 1 at the top to 9 at the bottom. The columns are numbered from 1 at the left to 9 at the right. The input contains exactly one constraint per cage.

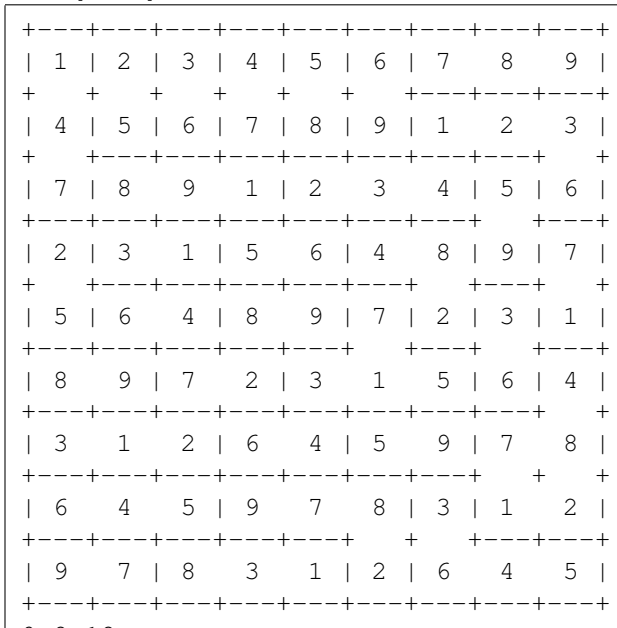
Output

Display `OK` if all of the constraints are met. Otherwise, display `NotOK` (note that there is no whitespace between `Not` and `OK`).

Notes

- Sample Input 1 is Figure K.2.
- Sample Input 2 is Figure K.1 with the centre entry changed to a 4. The cages correspond to the normal Sudoku constraints.

Sample Input 1



```
9 8 18
2 1 12
1 2 7
1 3 9
1 4 11
1 5 13
1 6 15
1 8 24
2 7 12
3 2 18
3 5 9
3 8 14
4 1 7
4 2 4
4 4 11
4 6 14
4 9 8
5 2 10
5 4 17
5 6 16
5 8 9
6 1 17
6 3 9
6 9 22
7 1 6
7 4 10
7 6 14
8 1 15
8 4 26
9 1 16
9 3 12
```

Sample Output 1

```
OK
```

Sample Input 2

```
+---+---+---+---+---+---+---+---+---+
| 2   9   5 | 7   4   3 | 8   6   1 |
+   +   +   +   +   +   +   +   +   +
| 4   3   1 | 8   6   5 | 9   2   7 |
+   +   +   +   +   +   +   +   +   +
| 8   7   6 | 1   9   2 | 5   4   3 |
+---+---+---+---+---+---+---+---+---+
| 3   8   7 | 4   5   9 | 2   1   6 |
+   +   +   +   +   +   +   +   +   +
| 6   1   2 | 3   4   7 | 4   9   5 |
+   +   +   +   +   +   +   +   +   +
| 5   4   9 | 2   1   6 | 7   3   8 |
+---+---+---+---+---+---+---+---+---+
| 7   6   3 | 5   2   4 | 1   8   9 |
+   +   +   +   +   +   +   +   +   +
| 9   2   8 | 6   7   1 | 3   5   4 |
+   +   +   +   +   +   +   +   +   +
| 1   5   4 | 9   3   8 | 6   7   2 |
+---+---+---+---+---+---+---+---+---+
2 2 45
5 2 45
8 2 45
2 5 45
5 5 45
8 5 45
2 8 45
5 8 45
8 8 45
```

Sample Output 2

```
NotOK
```

Problem L
Loading Cargo
Time limit: 3 seconds

We need to ship N capsules containing different potentially hazardous chemicals using an aircraft. The aircraft has one cargo compartment on the left with free space for L capsules and one cargo compartment on the right with free space for R capsules. According to the safety regulations, some capsules conflict with each other and cannot be loaded into the same cargo compartment (in case there is leakage and reaction). Given this information, your job is to determine whether all the capsules can be loaded into the aircraft.

Clip art by Chris Martin. Distributed under CC-BY-SA. Source: Wikimedia Commons

Input

The input starts with a line containing four integers L, R, N and C, where L ($0 \leq L \leq 2\,000$) is the capacity of the left cargo compartment, R ($0 \leq R \leq 2\,000$) is the capacity of the right cargo compartment, N ($0 \leq N \leq 2\,000$) is the number of capsules and C ($0 \leq C \leq 200\,000$) is the number of conflicts.

The next C lines describe the conflicts. Each of these lines contain two integers X ($0 \leq X \leq N-1$) and Y ($0 \leq Y \leq N-1$ and $X \neq Y$) representing a pair of capsules that conflict with each other. Each pair of capsules that conflict only appear once in the input. The capsules are numbered from 0 to $N-1$.

Output

Display `Yes` if all the capsules can be loaded into the aircraft and display `No` otherwise.

Sample Input 1

```
0 0 0 0
```

Sample Output 1

```
Yes
```

Sample Input 2

```
3 3 3 3
0 1
1 2
2 0
```

Sample Output 2

```
No
```

Sample Input 3

```
3 3 6 4
0 1
0 2
3 4
3 5
```

Sample Output 3

```
Yes
```

This page is intentionally left (almost) blank.