Problem A. Anti-Lines (Division 1 Only!)

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	256 mebibytes

In this task, we will tell about the classic version of the game "Lines". There is a square field of 9×9 cells. Each cell can be empty or contain a single ball of one of 7 colors. Each turn the computer puts three balls of random colors in random empty cells. After that player can move one ball in any empty cell. If 5 or more balls of the same color arranged in a vertical , horizontal or diagonal lines, these balls are destroyed, and the player gets points and an extra turn.

In this task, everything is a bit different: we hacked the computer and are now able to select empty cells on each turn, which will be occupied by new balls, and colors of these new balls. Two new balls appear on each turn. Player doesn't move anything (because we are directing the actions of the computer). Our goal — to make such moves, that allow us on the first destruction to destroy as many balls as possible. That is, you can make a few moves to build some successful design, and then destroy it in one move . You cannot make additional destructions to clear map.

Two balls are put in the field simultaneously. Thereafter, any ball that belongs to the vertical, horizontal or diagonal line of at least 5 balls of the same color, instantly collapses. If there are less than two empty cells, the next turn is not available.

Input

Input contains exactly 9 rows by 9 characters each — the current position in the game. Each character is either a digit from 1 to 7 , meaning the color of the ball in the current cell, or character '.' if the cell is empty. It is guaranteed that there is no line with 5 or more balls of the same color.

Output

The first line of output should contain two numbers D and M — maximum number of balls that can be destroyed by one stroke and the number of moves that lead to this destruction. $2M$ lines follow(2 lines on each turn). Each line should contain three numbers r_i, c_i and $color_i$ ($1 \le r_i, c_i \le 9$ $1 \le color_i \le 7$) — number of row and column of empty cells, where ball of $color_i$ color was placed. Both ball of one turn were placed at the same time. At the time of first destruction you must destroy exactly D balls (theoretically its possible to destroy them in a few times and make extra moves, but this is not recommended).

If no one ball can be destroyed, D must be 0. M does not exceed 100.

Examples

standard input	standard output
723134552 2421.2456 353...442 14...2..4 23.2....4 4....1..1 1222..112 3621..124 631211777	25 9 6 7 2 3 4 2 5 8 2 5 7 2 5 3 2 6 3 2 3 6 2 8 5 2 7 5 2 6 5 2 5 5 2 2 5 2 3 5 2 4 8 2 4 7 2 4 4 2 5 6 2 4 5 2

Problem B. Big Fellow (Division 1 Only!)

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

You got nm-nashki for your Birthday. It is basically a table consisting of n rows and m columns. In each cell, there is a card with a positive number. When all the cards are in their place, the table looks like this:

1	2	3	...	m
m+1	m+2	m+3	...	2m
2m+1	2m+2	2m+3	...	3m
...	...	...	...	...
(n-1)m+1	(n-1)m+2	(n-1)m+3	...	nm

Once you came home and saw that someone has played with your table. It now looks like this:

1	n+1	2n+1	...	(m-1)n+1
2	n+2	2n+2	...	(m-1)n+2
3	n+3	2n+3	...	(m-1)n+3
...	...	...	...	...
n	2n	3n	...	mn

You hated the way table looks now, and you decided to put all the cards in place. To do this, you decide to perform a sequence of operations. Each operation consists of the following:

1) Raise any card up in the air
2) Find its correct position in the table
3) If this place is available, put the card that was in the air on it
4) Otherwise, swap the card lying at this place, with the card in the air, and return to step 2.

However you decided to put yourself the condition that in the process of placing each of nm cards should appear in the air at least once. This means that even if the card is already on its place, you have to spend operation on a lifting and putting it back.

What is the minimum number of operations you have to perform to make your table look better?

Input

Single line contains two positive integers n and m, such that $n \cdot m \leq 10^{14}$.

Output

Print a single number - the minimum number of required operations.

Examples

standard input	standard output
4 4	10
2 3	3

Problem C. Concave Polygon (Division 1 Only!)

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	256 mebibytes

You have simple polygon without self-intersections or self-tangency. You need to cut it diagonally into the minimum number of convex polygons. As the edges of convex polygons, you can use only the vertices of the original polygon. No three vertices of any convex polygon You can use sides of the original polygon and its diagonals as sides of new convex polygons y. Each of the convex polygons must lie within the original polygon. The union of all convex polygons must be the original polygon. Also there shouldn't be two convex polygons with a positive area of intersection.

Input

The first line contains the number N ($3 \le N \le 150$) — the number of vertices of the original polygon. Each of the next N lines contains two integers x_i and y_i ($-10\,000 \le x_i, y_i \le 10\,000$) — coordinates of i-th vertex of the polygon in counter-clockwise order. It is guaranteed that the polygon is non-degenerate.

Output

Output one number — minimum number of convex polygons the original polygon can be cut into.

Examples

standard input	standard output
7 3 -1 3 1 2 1 3 3 0 0 3 -3 2 -1	3

Problem D. Dead Points (Division 1 Only!)

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

There are N points on a plane. Each of them is colored in one of two colors: black and white. Points are involved in such process: on each iteration any point which sees at least one point of another color dies. Point A sees point B, if the segment AB doesn't contain other alive points. During one iteration, all necessary points die simultaneously. If there are no alive points of any color, the process ends.

Determine how many iterations it requires to finish the process, and what color will survive if any.

Input

The first line contains number N ($1 \le N \le 100\,000$) — number of points on the plane. Next N lines contain three integers each: x_i, y_i and $color_i$ ($0 \le x_i, y_i \le 1\,000\,000$, $color_i \in \{0, 1\}$) — coordinates and color of i-th point. 0 means white, 1 means black. No two points have the same coordinates.

Output

Output the word "`Draw`", if all points will die, "`Black`", if black points will survive, or "`White`" if white will survive. In the same line output number of iterations (separated by space).

Examples

standard input	standard output
3 0 0 0 1 0 1 0 1 1	Draw 1
3 0 0 0 1 0 1 2 0 1	Black 1

Problem E. Equilateral Polygon

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

You want to build a convex polygon, with sides 1 unit long and with all angles at the vertices different.

Input

The only number N ($5 \le N \le 100$) — the number of vertices in the polygon.

Output

Exactly N lines. i-th line should contain two real numbers x_i and y_i (recommended with 16 digits after the decimal point) which are the coordinates of the i-th vertex of the polygon. x_i and y_i should not exceed 10^3 by absolute value. Vertices can be printed in clockwise or counterclockwise order.

The lengths of each side should not differ from 1 by more than 10^{-8}. Any two angles must differ by at least 10^{-4} radians. All angles have to be in the range of $[10^{-4}, \pi - 10^{-4}]$.

It is guaranteed that at least one polygon which satisfies the condition exists.

Examples

standard input	standard output
5	1.0000000000000000 1.0000000000000000 2.0000000000000000 1.0000000000000000 2.3093247548365476 1.9509564638012140 1.5003804226047008 2.5388417139563542 0.6912907818637410 1.9511564638048255

Problem F. Flawless Numbers

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	256 mebibytes

The number N is called flawless if $\frac{\sigma(N)}{N} = \frac{A}{B}$ where $\sigma(N)$ is the sum of all divisors of N. For given A and B find all flawless numbers from 1 to 10^{14} inclusive.

Input

Two positive integers A and B ($1 \le A, B \le 100$, A is an odd number (suddenly!), $2 \le \frac{A}{B} \le 5$).

Output

The first line should contain the integer K — the number of flawless numbers. Each flawless number has to appear exactly once. Numbers can be printed in any order. It is guaranteed that K does not exceed 1000.

Examples

standard input	standard output
5 2	3 24 91963648 10200236032

Problem G. Grep

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

You have a string S of length N with symbols ‘a‘, ‘b‘ and ‘c‘ and string T of length N with symbols ‘.‘ and ‘*‘. In one operation you can cyclically shift string S by one character to the right or left. You need to shift the string S in such a way that for all positions in string T which contain ‘ texttt*‘ each of the letters ‘ texttta‘, ‘ texttt b‘ and ‘ texttt c‘ was in this position at least once. Find the minimum number of operations needed to ensure this fact.

Input

The first line contains the string S, which consists entirely of characters ‘a‘, ‘b‘and ‘c‘. There is at least one characters of each of ‘a‘, ‘b‘ and ‘c‘ in the string S. The second line contains the string T, consisting only of characters ‘.‘ and ‘*‘. $3 \le |S| = |T| \le 1\,000\,000$.

Output

Output one number — minimum number of shifts.

Examples

standard input	standard output
abacaba ...**..	3

Problem H. Hash It!

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

Initially there is an empty string S. Two types of operations are applied to it:

- Add symbol c to the right of S.
- Remove the last character from the string S.

Output the number of different substrings in S after each operation.

Input

One line Q ($1 \le |Q| \le 100\,000$), that represents the sequence of operations with a string S. If i-th character of a string Q is a lowercase English letter, then as i-th operation we add that letter to the end of line S, if i-th character of a string Q is '-', then as the i-th operation we remove the last character from the string S. It is guaranteed that before the remove operation string S is not empty.

Output

Print exactly $|Q|$ lines. i-th line should contain one number — the number of different substrings in S after applying the first i operations.

Examples

standard input	standard output
aba-caba	1 3 5 3 6 9 12 17

Problem I. Interactive Problem 2

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	**8 mebibytes**

Attention! This problem has a special memory limit.

There is a rooted binary tree. All tree nodes are numbered from 1 to N. Each node can have either 2 or 0 children. You have to traverse the tree by breadth first search or depth first search — at your choice, and print the numbers of vertices in the order you preferred, and the sum of distances from the root to all the vertices. Root always has number 1.

If you output the vertices in the order of breadth first search, then you should output the numbers of vertices in the same layer from leftmost to rightmost. If you output the vertices in the order of depth first search, the numbers of vertices should be printed at the moment of the first entrance, and select the left child before the right one while traversing.

More formally: to each vertex v_i assign the string S_i, where the j-th symbol is 'l', if j-th edge on the path from the root to the vertex v_i directs to the left child and 'r' if to the right one. Number of characters in line S_i is equal to the distance from the vertex to the root.

When traversing in breadth first search order vertex v_i should be visited before vertex v_j, if and only if:

- S_i is smaller than S_j, if S_i and S_j have different lengths
- S_i is lexicographically smaller than S_j, if S_i and S_j have same lengths

When traversing in depth first search order vertex v_i should be visited before vertex v_j, if and only if S_i is lexicographically smaller than S_j.

Input

The first line contains integer N ($1 \le N \le 1\,000\,000$) — the number of vertices in the tree. Each of the next N lines contains a pair of numbers l_i, r_i ($1 \le l_i, r_i \le N$) — the numbers of the left and right children of the i-th vertex, or two zeros, if the vertex i is a leaf.

Output

On the first line print "`DFS`", if you chose the depth first search, or "`BFS`", if you chose the breadth first search. In the following N lines print the numbers of vertices in the chosen order of traversal. In the last line print the sum of the distances from the root to all vertices.

Examples

standard input	standard output
5 3 4 0 0 0 0 2 5 0 0	DFS 1 3 4 2 5 6

Problem J. Jolly Dolls

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 seconds
Memory limit:	256 mebibytes

Matryoshkas are wooden dolls nested one inside the other. Each doll has its own external volume out_i and volume in_i of the empty space inside it. One doll can be embedded into another, if the external volume of the first one is smaller (here you ask yourself "smaller???"Yes, you were not confused. Exactly smaller) than the empty space inside the second. We can directly put no more than one doll inside the other. But, this nested doll can contain another doll and so on. That is, if the two dolls are actually located inside the third doll, than one of them is located inside the other.

Matryoshkas are nested optimally if the total volume of empty space inside all matryoshkas is minimal. Find the number of ways to optimally place matryoshkas into each other.

Input

The first line contains integer N ($1 \le N \le 100\,000$) — the number of matryoshkas in set. The i-th of the following N lines contains two integers out_i and in_i ($1 \le in_i < out_i \le 1\,000\,000\,000$) — external volume and the volume of empty space inside the i-th matryoshka.

Output

The number of optimal placements of nesting dolls modulo 1 000 000 007.

Examples

standard input	standard output
3 5 4 4 2 3 2	1