

Problem E. Equilateral Polygon

Input file: **stdin**
Output file: **stdout**
Time limit: 1 s
Memory limit: 256 Mb

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 ($5 \leq N \leq 100$) — 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

stdin	stdout
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: `stdin`
Output file: `stdout`
Time limit: 5 s
Memory limit: 256 Mb

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 ($1 \leq A, B \leq 100$, A is an odd number (suddenly!), $2 \leq \frac{A}{B} \leq 5$).

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

stdin	stdout
5 2	3 24 91963648 10200236032

Problem G. Grep

Input file: `stdin`
Output file: `stdout`
Time limit: 1 s
Memory limit: 256 Mb

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 '*'. $3 \leq |S| = |T| \leq 1\,000\,000$.

Output

Output one number — minimum number of shifts.

Examples

stdin	stdout
abacaba ...**..	3

Problem H. Hash It!

Input file: `stdin`
Output file: `stdout`
Time limit: 1 s
Memory limit: 256 Mb

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 ($1 \leq |Q| \leq 100\,000$), 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

stdin	stdout
aba-caba	1
	3
	5
	3
	6
	9
	12
	17

Problem I. Interactive Problem 2

Input file: `stdin`
Output file: `stdout`
Time limit: `1 s`
Memory limit: `10 Mb`

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 ($1 \leq N \leq 1\,000\,000$) — the number of vertices in the tree. Each of the next N lines contains a pair of numbers l_i, r_i ($1 \leq l_i, r_i \leq N$) — 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

stdin	stdout
5	DFS
3 4	1
0 0	3
0 0	4
2 5	2
0 0	5
	6

Problem J. Jolly Dolls

Input file: `stdin`
Output file: `stdout`
Time limit: 1 s
Memory limit: 256 Mb

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 ($1 \leq N \leq 100\,000$) — the number of matryoshkas in set. The i -th of the following N lines contains two integers out_i and in_i ($1 \leq in_i < out_i \leq 1\,000\,000\,000$) — 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

stdin	stdout
3 5 4 4 2 3 2	1

Problem K. Block Shuffling (junior league statements)

Input file: `stdin`
Output file: `stdout`
Time limit: `1 s`
Memory limit: `256 Mb`

There is the identity permutation of 2^p elements $(1, 2, 3, \dots, 2^p)$. You have to perform the following sequence of operations on it:

Break permutation down into continuous nonintersecting blocks 2^k long each. In other words, elements in positions $(1, \dots, 2^k)$ form the first block, elements in positions $(2^k + 1, \dots, 2 \cdot 2^k)$ form the second block, elements $(2 \cdot 2^k + 1, \dots, 3 \cdot 2^k)$ form the third one and so on. Then swap the neighboring blocks. That is, the first block is swapped with the second one, the third one with the fourth one and so on.

Given a set of positions, you have to tell which elements of the permutation are in these positions after all operations.

Input

First line contains three numbers N , K and Q ($N = 2^p$, $0 \leq p \leq 21$, $1 \leq K, Q \leq 100\,000$) — number of elements in permutation, the number of operations to perform and the number of positions we are interested in.

It is followed by K lines each having one number k_i ($0 \leq k_i < p$), meaning that you need to swap the neighboring blocks of 2^{k_i} elements.

Q lines follow, each having one number q_i ($1 \leq q_i \leq N$) — the numbers of positions we are interested in.

Output

Output Q lines, in the i -th of which there is a number of element of permutation in position q_i .

Examples

<code>stdin</code>	<code>stdout</code>
8 3 3	8
0	6
1	2
2	
1	
3	
7	

Problem L. Digit Permutation (junior league statements)

Input file: **stdin**
Output file: **stdout**
Time limit: **1 s**
Memory limit: **256 Mb**

Find the number in the numeric system with base B , satisfying the following restrictions:

- Each digit from 1 to $B - 1$ occurs exactly once (there are no zeroes).
- For any i : $1 \leq i \leq B - 1$, if you take first i digits of this number without changing the order, then the resulting number would be divisible by i .

Input

Number B ($2 \leq B \leq 36$) — base of numeric system.

Output

If required number does not exist, output “No”, otherwise output a single number in the numeric system with base B . Digits from 0 to 9 are defined by characters ‘0’ - ‘9’. Digits from 10 to 35 are defined by characters ‘A’ - ‘Z’. If there are several answers, output any of them.

Examples

stdin	stdout
10	381654729
18	No

Problem M. Mortal Points (junior league statements)

Input file: `stdin`
Output file: `stdout`
Time limit: 1 s
Memory limit: 256 Mb

There are N points on a line. 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 ($1 \leq N \leq 100\,000$) — number of points on the line. Next N lines contain two integers each: x_i and $color_i$ ($0 \leq x_i \leq 1\,000\,000$, $color_i \in \{0, 1\}$) — coordinate and color of the i -th point. 0 means white, 1 means black. None of two points have the same coordinate.

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

stdin	stdout
3 0 0 1 1 2 1	Black 1

Problem N. Non-convex Polygon (junior league statements)

Input file: `stdin`
Output file: `stdout`
Time limit: `1 s`
Memory limit: `256 Mb`

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 ($3 \leq N \leq 15$) — the number of vertices of the original polygon. Each of the next N lines contains two integers x_i and y_i ($-10\,000 \leq x_i, y_i \leq 10\,000$) — 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

stdin	stdout
7 3 -1 3 1 2 1 3 3 0 0 3 -3 2 -1	3