Problem B. Bubble Sort

Input file:	bubble.in
Output file:	bubble.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Bubble sort is a well known sorting algorithm that can be described by the following pseudocode:

```
define BubbleSort(a: array [1..n]):
  for i from 1 to n − 1:
    for j from 1 to n − 1:
      if a[j] > a[j + 1]:
        swap(a[j], a[j + 1])
```

Let us call one iteration of the external cycle a *bubble pass*.

```
define BubblePass(a: array [1..n]):
  for j from 1 to n − 1:
    if a[j] > a[j + 1]:
      swap(a[j], a[j + 1])
```

Actually, not all of $n - 1$ bubble passes in the original algorithm are always necessary. Some arrays get sorted after fewer passes. For example, array $\langle 2, 1, 4, 3 \rangle$ is sorted by just one bubble pass.

Actually there are many arrays that can be sorted by just one bubble pass. For example, 4 out of 6 permutations of numbers from 1 to 3 are sorted by a single bubble pass. Let us denote the number of permutations of numbers from 1 to n that are sorted by a single bubble pass as $B(n)$.

Given n and k you have to find the k-th lexicographically permutation of numbers from 1 to n that is sorted by a single bubble pass.

Input

The input file contains several test cases.

Each test case consists of two integers n and k on a line ($1 \le n \le 100\,000$, $1 \le k \le 5 \cdot 10^{18}$, k doesn't exceed $B(n)$).

The last line of the input file contains two zeroes, it must not be processed.

The sum of n for all test cases in the input file doesn't exceed 10^6.

Output

For each test case in the input file output one line that contains n integers — the k-th lexicographically permutation of numbers from 1 to n that is sorted by a single bubble pass.

Examples

bubble.in	bubble.out
3 1 3 2 3 3 3 4 0 0	1 2 3 1 3 2 2 1 3 3 1 2

Problem B. Antipalindromic Numbers

Input file:	`anti.in`
Output file:	`anti.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Palindrome is a word that is read the same way in either direction. Similarly let us define *antipalindrome* as a word that has no equal characters on the same positions when read from front to back and from back to front. For example "computer" is antipalindrome, but "information" is not ('m' is on the 6-th position in both "information" and "noitamrofni").

A number is called antipalindromic if its decimal notation is antipalindrome.

You are given a number x. Find the smallest antipalindromic number greater than x.

Input

Input file contains several test cases. Each test case is the number x ($1 \le x \le 10^{100}$). The last test case is followed by 0, it must not be processed.

Output

For each number in the input print the smallest antipalindromic number greater than it on a line by itself.

Examples

anti.in	anti.out
5 20 99 0	10 21 1010

Problem E. Chipmunks

Input file:	`chipmunks.in`
Output file:	`chipmunks.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Andrew and Ann work in a medical laboratory. They make tests of new medicines on chipmunks. Initially there were two chipmunks in the laboratory, and they were numbered 1 and 2. It was very convenient — you can say "chipmunks with total number 2" and understand that this is about chipmunk 2, or "chipmunks with total number 3" and understand that this is about both chipmunks.

However, soon the third chipmunk was bought for the laboratory. Now when you say "chipmunks with total number 3" it is not clear whether this is about chipmunk 3 or chipmunks 1 and 2. However, Andrew and Ann found the solution. Now they say "1 chipmunk with total number 3" or "2 chipmunks with total number 3" and it is clear what set of chipmunks is referred.

But the problems came when the fourth chipmunk was bought. If it were assigned number 4, the phrase "2 chipmunks with total number 5" would be ambiguous. So Andrew and Ann had to find another solution, and of course it was found. The chipmunk was assigned number 5. And again specifying the number of chipmunks in the set and their total number uniquely identified the set of chipmunks.

Help Andrew and Ann to continue assigning numbers to chipmunks as more and more of them are bought. Let first $n-1$ chipmunks be already assigned numbers in the following way: each new chipmunk is assigned the smallest possible positive integer number such that the number of chipmunks in a set and sum of their numbers uniquely identifies the set. Suppose the n-th chipmunk is bought. Find out what is the smallest possible number that can be assigned to the n-th chipmunk so that the above property was preserved.

Input

The input file contains one integer number n ($1 \le n \le 20$).

Output

Output one number — the number that must be assigned to the n-th chipmunk.

Examples

chipmunks.in	chipmunks.out
1	1
4	5
5	8

Problem C. Dowry

Input file:	`dowry.in`
Output file:	`dowry.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

The daughter of the King of Flatland is going to marry the beautiful prince. The prince is going to give the generous dowry for the King's daughter, but he is unsure which jewels from his collection to give.

There are n jewels in the collection of the prince, each jewel is characterized by its weight w_i and its value v_i. Prince would like to give as valuable dowry to the King as possible. But the King is wise and he would accept the jewels only if their total weight doesn't exceed R. On the other side the prince would consider himself greedy for the rest of his life if he gave the jewels with the total weight less than L.

Help the prince to choose jewels from his collection so that their total weight was between L and R (inclusive), and the total value of the selected jewels was maximal possible.

Input

The first line of the input file contains n ($1 \le n \le 32$), L and R ($0 \le L \le R \le 10^{18}$). The following n lines describe jewels and contain two numbers each — the weight and the value of the corresponding jewel ($1 \le w_i, v_i \le 10^{15}$).

Output

The first line of the output file must contain k — the number of jewels to present to the king. The second line must contain k integer numbers — the numbers of jewels. Jewels are numbered from 1 to n in order they are given in the input file.

If it is impossible to choose the jewels, output 0 at the first line of the output file.

Example

dowry.in	dowry.out
3 6 8 3 10 7 3 8 2	1 2

Problem E. Cryptography

Input file:	`crypto.in`
Output file:	`crypto.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Professor Klever is working on cracking the new cipher *Formally Incrackable Generaly Virtual Applied Masking* (FIGVAM). After investigating the cipher he reduced the problem of restoring the encryption key to the following problem: find two numbers x and y ($1 \le x, y \le n$) such that:

- $x \wedge y = y$;
- $(ax + by) \oplus (ay + bx)$ is maximal.

Here $p \wedge q$ means bitwise "and" of p and q, and $p \oplus q$ means bitwise "exclusive or" of p and q.

You are given n, a and b. Help professor to find x and y.

Input

Input file contains n, a and b ($1 \le n \le 100\,000$, $0 \le a, b \le 2000$).

Output

Output x and y that satisfy the given conditions.

Example

crypto.in	crypto.out
20 2 3	15 10

Problem E. Permutation Reconstruction

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

The famous Ulam Conjecture claims that if we take a graph G and consider a multiset G_{min} of graphs that are obtained from G by removing one of its vertices and all edges incident to it, then the graph G can be reconstructed from G_{min}. The Ulam conjecture is still not proven and no counterexample is known.

In this problem we will consider a similar problem for permutations. Consider a permutation $a = \langle a_1, a_2, \ldots a_n \rangle$ of numbers from 1 to n. Let us denote as a/i the permutation of $n-1$ numbers obtained from a by removing a number i and decreasing all numbers greater than i by one.

For example, if $a = \langle 1, 3, 5, 2, 6, 4 \rangle$ then $a/2 = \langle 1, 2, 4, 5, 3 \rangle$.

You are given $a/1, a/2, \ldots, a/n$ in some arbitrary order. You must restore the original permutation a.

Input

The first line of the input file contains n — the order of the initial permutation ($5 \le n \le 300$). The following n lines contain $n-1$ numbers each and specify a/i for all i in some order.

Output

Output n integer numbers — the permutation a. It is guaranteed that such permutation exists.

Example

permutation.in	permutation.out
6 1 3 5 2 4 1 3 4 2 5 1 2 4 5 3 2 4 1 5 3 1 4 2 5 3 1 3 2 5 4	1 3 5 2 6 4

In this example the factors are given in the following order: $a/6$, $a/4$, $a/2$, $a/1$, $a/3$ and $a/5$.

Problem F. Decoding Prefix Codes

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

A code is a mapping $c : \Sigma \to \Gamma^*$ of characters of the given alphabet Σ to words of some another alphabet Γ. In this problem Σ consists of lowercase letters of the English alphabet, and $\Gamma = \{0, 1\}$.

The code is called *prefix-free* or simply *prefix*, if no code word is a prefix of another word. For example, the code $c(\text{'a'}) = 00$, $c(\text{'b'}) = 01$, $c(\text{'c'}) = 1$ is prefix, but the code $c(\text{'a'}) = 0$, $c(\text{'b'}) = 01$ is not.

You are given a text and a string that is obtained from it by encoding it with some prefix code. You must restore the code that was used to encode the text. If there are several possible variants, output any one.

Input

The first line of the input file contains the given text. Its length does not exceed 1000. It contains only letters '`a`'–'`z`' of the English alphabet. There are at most 10 different characters in the given text.

The second line contains the encoded version of the given text. It is guaranteed that it is obtained from the given text by encoding by some prefix code. No word in the code used has length exceeding 10.

Output

Output the prefix code that could be used to encode the text to get the given encoded version. The number of lines must be equal to the number of different characters in the given text. Each line must contain a character followed by a space and the code word for this character.

Example

prefix.in	prefix.out
hello 0100111000	e 001 h 01 l 1 o 000

Problem C. Yellow Code

Input file:	`code.in`
Output file:	`code.out`
Time limit:	1 second
Memory limit:	64 megabytes

Inspired by Gray code, professor John Yellow has invented his own code. Unlike in Gray code, in Yellow code the adjacent words have many different bits.

More precisely, for $s = 2^n$ we call the permutation $a_1, a_2, \ldots, a_s$ of all n-bit binary words the *Yellow code* if for all $1 \le k < s$ words a_k and a_{k+1} have at least $\lfloor n/2 \rfloor$ different bits and a_1 and a_s also have at least $\lfloor n/2 \rfloor$ different bits.

Given n you have to find the n-bit Yellow code or detect that there is none.

Input

Input file contains the number n ($2 \le n \le 12$).

Output

Output 2^n n-bit binary vectors in the order they appear in some n-bit Yellow code, one on a line. If there is no n-bit Yellow code, output "`none`" on the first line of the output file.

Example

code.in	code.out
4	0000 1111 0001 1110 0010 1101 0011 1100 0101 1011 0100 1010 0110 1000 0111 1001

Problem D. Matrix Multiplication

Input file:	`matrix.in`
Output file:	`matrix.out`
Time limit:	1 second

Let us consider undirected graph $G = \langle V, E \rangle$ which has N vertices and M edges. Incidence matrix of this graph is $N \times M$ matrix $A = \{a_{ij}\}$, such that a_{ij} is 1 if i-th vertex is one of the ends of j-th edge and 0 in the other case. Your task is to find the sum of all elements of the matrix $A^T A$.

Input

The first line of the input file contains two integer numbers — N and M ($2 \le N \le 10\,000$, $1 \le M \le 100\,000$). $2M$ integer numbers follow, forming M pairs, each pair describes one edge of the graph. All edges are different and there are no loops (i.e. edge ends are distinct).

Output

Output the only number — the sum requested.

Example

matrix.in	matrix.out
4 4 1 2 1 3 2 3 2 4	18

Problem H. Saving Princess

Input file:	princess.in
Output file:	princess.out
Time limit:	1 second
Memory limit:	64 megabytes

Saving princesses is always a hard work. Ivan D'Ourack is planning to save the princess locked in the tower. However, n dangerous monsters are guarding the road from the city where Ivan lives to the tower where the princess is locked.

Fortunately Ivan is a warrior and a magician. Thus he can defeat monsters in a fight, and enchant them to pass unnoticed.

Initially Ivan has h health points, strength s, spell power p and m mana points. To defeat i-th monster in a fight, he must have strength at least s_i, and he loses $\max(2s_i - s, 0)$ health points in a fight. If the number of health points becomes 0 or less, Ivan dies. After defeating a monster Ivan's strength increases by 1.

To enchant i-th monster Ivan must have spell power at least p_i and he spends m_i mana points to do it. If Ivan does not have m_i mana points, he cannot enchant the monster. After enchanting the monster Ivan's spell power increases by 1.

Find out, whether Ivan can save princess, and if he can how to do it.

Input

The first line of the input file contains n, h, s, p and m ($1 \le n \le 50$, $1 \le h \le 50$, $0 \le s, p, m \le 50$). The following n lines contain three integer numbers each — s_i, p_i, and m_i ($1 \le s_i, p_i, m_i \le 50$).

Output

If Ivan cannot save princess, output "UNLUCKY". In the other case output n characters, the i-th character must be 'D' if Ivan must defeat the i-the monster, or 'E' if he must enchant it.

Example

princess.in	princess.out
3 12 5 5 6 5 5 2 6 5 2 6 7 3	DED
3 11 5 5 6 5 5 2 6 5 2 6 7 3	UNLUCKY

Problem J. Tree Analysis

Input file:	`tree.in`
Output file:	`tree.out`
Time limit:	1 second
Memory limit:	64 megabytes

A *rooted tree* is a directed graph with one selected node r, called *root*, that satisfied the following conditions:

1. all nodes are reachable from r;
2. r has no edges entering it;
3. each node u except r has exactly one edge entering it, the source of this edge is called the *parent* of u.

Each node u in a rooted tree defines a set $T(u)$ of nodes, reachable from it. The graph induced by $T(u)$ is also a rooted tree, it is called a subtree rooted in u.

Two graphs G_1 and G_2 are called *isomorphic* if there is a bijection φ between their sets of vertices, such that $u \to v$ is an edge in G_1 if and only if $\varphi(u) \to \varphi(v)$ is an edge in G_2. Two rooted trees are isomorphic if they are isomorphic as graphs.

One way to check whether the two trees are isomorphic uses *subtree isomorphism relation*. Two nodes u and v of the tree are called *subtree isomorphic* if subtrees rooted in u and v respectively are isomorphic. Clearly, subtree isomorphism relation is an equivalence relation. Therefore, all nodes of the tree can be divided into equivalence classes by this relation.

Given a rooted tree, find its subtree isomorphism relation equivalence classes.

Input

The first line of the input file contains n — the number of nodes in the tree ($1 \le n \le 100\,000$). Let nodes be numbered in such a way that the root has the number 1 and for each other vertex its number is greater than the number of its parent.

The rest of the file contains $n - 1$ integer numbers separated by spaces and line feeds. The i-th of these numbers is the parent of the $(i + 1)$-th node.

Output

Output n integer numbers — for each node output some integer number ranging from 1 to n. The numbers for any two nodes must be equal if and only if these nodes are subtree isomorphic.

Example

tree.in	tree.out
9 1 2 2 1 5 5 7 7	1 2 3 3 4 3 2 3 3

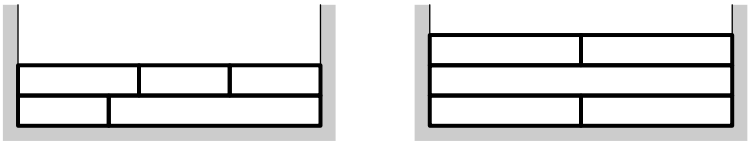
Problem I. Crazy Wall

Input file:	`wall.in`
Output file:	`wall.out`
Time limit:	1 second
Memory limit:	64 megabytes

The authors of the game "*Crazy Tetris*" decided to create the new game, they called "*Crazy Wall*". The goal of the game is to build a wall to lock up the pass. The wall must be as high as possible.

The width of the pass is w. The wall must be built of several bricks. The bricks fall from above one after another. For each brick the positions of its left and right borders are known. The player is not allowed to move the bricks, but she is allowed to choose the order in which the bricks fall. She is also allowed to stop the bricks and confirm that the wall is completed.

The wall consists of several rows, the width of each row is w. Additionally, the wall must be *solid*. The wall is called solid, if no two bricks have the same position of the left border, except at the left side of the pass, nor any two bricks have the same position of the right border, except at the right side of the pass. For example, the wall on the left on the picture below is solid, but the wall on the right is not.



Given the description of the bricks, find out what is the maximal possible height of the solid wall one can build, and what bricks must be used to build it.

Input

The first line of the input file contains n and w — the number of bricks and the width of the pass, respectively ($1 \le n \le 2000$, $1 \le w \le 2 \cdot 10^9$). The following n lines describe bricks. Each brick is described with two integer numbers l_i and r_i — the position of its left and right border ($0 \le l_i < r_i \le w$).

Output

At the first line of the output file print h — the maximal possible height of the wall. At the following h lines describe the rows. Each row description must consist of numbers of bricks used to create the row. List bricks that the row consists of from left to right.

Examples

wall.in	wall.out
6 10 0 3 3 7 7 10 3 10 0 4 4 7	2 1 4 5 6 3
5 10 0 5 0 5 0 10 5 10 5 10	2 1 4 3

Problem H. Subword

Input file:	`subword.in`
Output file:	`subword.out`
Time limit:	1 second
Memory limit:	64 megabytes

Let us introduce the free group with two non-commuting generators.

We will denote generators as a and b and their reverses as A and B respectively. The group can be described as a set of words consisting of a, b, A and B, the operation of concatentation, and the generating relations $aA = Aa = Bb = bB = \varepsilon$.

In other words, we say that two words are equivalent if one can be transformed to another by successively performing the following operations:

- remove a subword equal to Aa, aA, Bb or bB from any position;
- add a subword equal to Aa, aA, Bb or bB to any position.

For example, words $abAaBBabbA$ and $aAaBabaAbA$ are equivalent:

$$abAaBBabbA \to abBBabbA \to aBabbA \to aAaBabbA \to aAaBabaAbA,$$

but words $abAB$ and $baBA$ are not.

Equivalence classes of words are exactly the elements of the group.

It is interesting to note that for each word α and each word β there is such word γ equivalent to α that contains β as a subword. Your task in this problem is to find the shortest such word.

Input

The first line of the input file contains α. The second line contains β. Both words are not empty and contain at most 2000 characters each.

Output

Output one word — the shortest word equivalent to α that contains β as a subword. If there are several solutions, print any one.

Examples

subword.in	subword.out
abAaBBabbA AaBaba	aAaBabaAbA