A. Modular Exponentiation

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The following problem is well-known: given integers n and m, calculate

$$2^n \bmod m,$$

where $2^n = 2 \cdot 2 \cdot \ldots \cdot 2$ (n factors), and $x \bmod y$ denotes the remainder of division of x by y.

You are asked to solve the "reverse" problem. Given integers n and m, calculate

$$m \bmod 2^n.$$

Input

The first line contains a single integer n ($1 \le n \le 10^8$).

The second line contains a single integer m ($1 \le m \le 10^8$).

Output

Output a single integer — the value of $m \bmod 2^n$.

Examples

input

```
4
42
```

output

```
10
```

input

```
1
58
```

output

```
0
```

input

```
98765432
23456789
```

output

```
23456789
```

Note

In the first example, the remainder of division of 42 by $2^4 = 16$ is equal to 10.

In the second example, 58 is divisible by $2^1 = 2$ without remainder, and the answer is 0.

B. Christmas Spruce

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Consider a rooted tree. A rooted tree has one special vertex called the root. All edges are directed from the root. Vertex u is called a *child* of vertex v and vertex v is called a *parent* of vertex u if there exists a directed edge from v to u. A vertex is called a *leaf* if it doesn't have children and has a parent.

Let's call a rooted tree a *spruce* if its every non-leaf vertex has at least 3 leaf children. You are given a rooted tree, check whether it's a spruce.

The definition of a rooted tree can be found here.

Input

The first line contains one integer n — the number of vertices in the tree ($3 \le n \le 1\,000$). Each of the next $n - 1$ lines contains one integer p_i ($1 \le i \le n - 1$) — the index of the parent of the $i + 1$-th vertex ($1 \le p_i \le i$).

Vertex 1 is the root. It's guaranteed that the root has at least 2 children.

Output

Print "`Yes`" if the tree is a spruce and "`No`" otherwise.

Examples

input

```
4
1
1
1
```

output

```
Yes
```

input

```
7
1
1
1
2
2
2
```

output

```
No
```

input

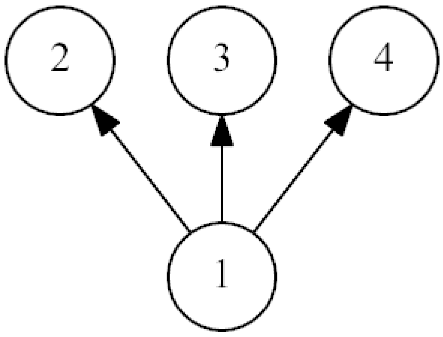
```
8
1
1
1
1
3
3
3
```

output

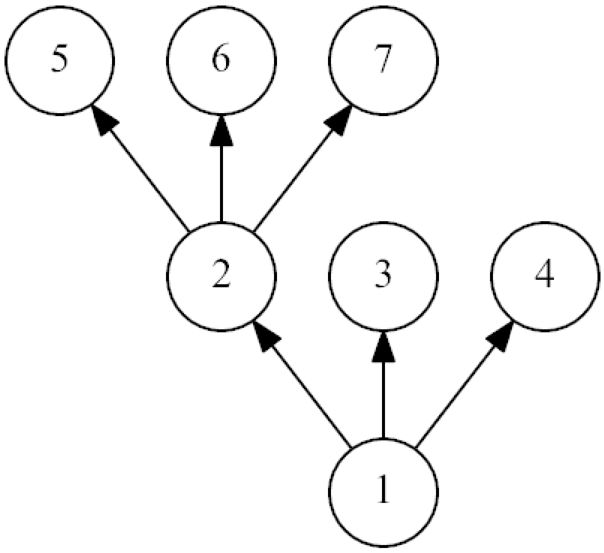
```
Yes
```

Note

The first example:

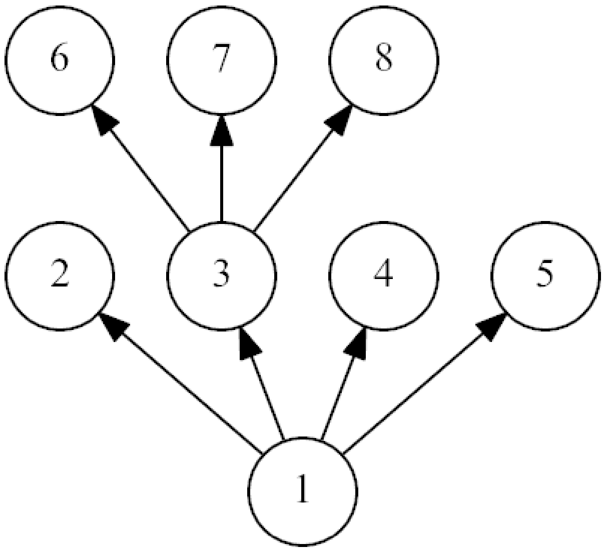


The second example:



It is not a spruce, because the non-leaf vertex 1 has only 2 leaf children.

The third example:


6
7
8
2
3
4
5
1

C. Party Lemonade

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

A New Year party is not a New Year party without lemonade! As usual, you are expecting a lot of guests, and buying lemonade has already become a pleasant necessity.

Your favorite store sells lemonade in bottles of n different volumes at different costs. A single bottle of type i has volume 2^{i-1} liters and costs c_i roubles. The number of bottles of each type in the store can be considered infinite.

You want to buy at least L liters of lemonade. How many roubles do you have to spend?

Input

The first line contains two integers n and L ($1 \le n \le 30$; $1 \le L \le 10^9$) — the number of types of bottles in the store and the required amount of lemonade in liters, respectively.

The second line contains n integers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le 10^9$) — the costs of bottles of different types.

Output

Output a single integer — the smallest number of roubles you have to pay in order to buy at least L liters of lemonade.

Examples

input

```
4 12
20 30 70 90
```

output

```
150
```

input

```
4 3
10000 1000 100 10
```

output

```
10
```

input

```
4 3
10 100 1000 10000
```

output

```
30
```

input

```
5 787787787
123456789 234567890 345678901 456789012 987654321
```

output

```
44981600785557577
```

Note

In the first example you should buy one 8-liter bottle for 90 roubles and two 2-liter bottles for 30 roubles each. In total you'll get 12 liters of lemonade for just 150 roubles.

In the second example, even though you need only 3 liters, it's cheaper to buy a single 8-liter bottle for 10 roubles.

In the third example it's best to buy three 1-liter bottles for 10 roubles each, getting three liters for 30 roubles.

D. Too Easy Problems

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are preparing for an exam on scheduling theory. The exam will last for exactly T milliseconds and will consist of n problems. You can either solve problem i in exactly t_i milliseconds or ignore it and spend no time. You don't need time to rest after solving a problem, either.

Unfortunately, your teacher considers some of the problems too easy for you. Thus, he assigned an integer a_i to every problem i meaning that the problem i can bring you a point to the final score only in case you have solved no more than a_i problems overall (including problem i).

Formally, suppose you solve problems $p_1, p_2, \ldots, p_k$ during the exam. Then, your final score s will be equal to the number of values of j between 1 and k such that $k \le a_{p_j}$.

You have guessed that the real first problem of the exam is already in front of you. Therefore, you want to choose a set of problems to solve during the exam maximizing your final score in advance. Don't forget that the exam is limited in time, and you must have enough time to solve all chosen problems. If there exist different sets of problems leading to the maximum final score, any of them will do.

Input

The first line contains two integers n and T ($1 \le n \le 2 \cdot 10^5$; $1 \le T \le 10^9$) — the number of problems in the exam and the length of the exam in milliseconds, respectively.

Each of the next n lines contains two integers a_i and t_i ($1 \le a_i \le n$; $1 \le t_i \le 10^4$). The problems are numbered from 1 to n.

Output

In the first line, output a single integer s — your maximum possible final score.

In the second line, output a single integer k ($0 \le k \le n$) — the number of problems you should solve.

In the third line, output k distinct integers $p_1, p_2, \ldots, p_k$ ($1 \le p_i \le n$) — the indexes of problems you should solve, in any order.

If there are several optimal sets of problems, you may output any of them.

Examples

input

```
5 300
3 100
4 150
4 80
2 90
2 300
```

output

```
2
3
3 1 4
```

input

```
2 100
1 787
2 788
```

output

```
0
0
```

input

```
2 100
2 42
2 58
```

output

```
2
2
1 2
```

Note

In the first example, you should solve problems 3, 1, and 4. In this case you'll spend $80 + 100 + 90 = 270$ milliseconds, falling within the length of the exam, 300 milliseconds (and even leaving yourself 30 milliseconds to have a rest). Problems 3 and 1 will bring you a point each, while problem 4 won't. You'll score two points.

In the second example, the length of the exam is catastrophically not enough to solve even a single problem.

In the third example, you have just enough time to solve both problems in $42 + 58 = 100$ milliseconds and hand your solutions to the teacher with a smile.

E. Logical Expression

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a boolean function of three variables which is defined by its truth table. You need to find an expression of minimum length that equals to this function. The expression may consist of:

- Operation `AND` ('`&`', ASCII code 38)
- Operation `OR` ('`|`', ASCII code 124)
- Operation `NOT` ('`!`', ASCII code 33)
- Variables `x`, `y` and `z` (ASCII codes 120-122)
- Parentheses ('`(`', ASCII code 40, and '`)`', ASCII code 41)

If more than one expression of minimum length exists, you should find the lexicographically smallest one.

Operations have standard priority. `NOT` has the highest priority, then `AND` goes, and `OR` has the lowest priority. The expression should satisfy the following grammar:

E ::= E '`|`' T | T

T ::= T '`&`' F | F

F ::= '`!`' F | '`(`' E '`)`' | '`x`' | '`y`' | '`z`'

Input

The first line contains one integer n — the number of functions in the input ($1 \le n \le 10\,000$).

The following n lines contain descriptions of functions, the i-th of them contains a string of length 8 that consists of digits `0` and `1` — the truth table of the i-th function. The digit on position j ($0 \le j < 8$) equals to the value of the function in case of $x = \lfloor \frac{j}{4} \rfloor$, $y = \lfloor \frac{j}{2} \rfloor \bmod 2$ and $z = j \bmod 2$.

Output

You should output n lines, the i-th line should contain the expression of minimum length which equals to the i-th function. If there is more than one such expression, output the lexicographically smallest of them. Expressions should satisfy the given grammar and shouldn't contain white spaces.

Example

input

```
4
00110011
00000111
11110000
00011111
```

output

```
y
(y|z)&x
!x
x|y&z
```

Note

The truth table for the second function:

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

F. Strongly Connected Tournament

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There is a chess tournament in All-Right-City. n players were invited to take part in the competition. The tournament is held by the following rules:

1. Initially, each player plays one game with every other player. There are no ties;
2. After that, the organizers build a complete directed graph with players as vertices. For every pair of players there is exactly one directed edge between them: the winner of their game is the startpoint of this edge and the loser is the endpoint;
3. After that, the organizers build a condensation of this graph. The condensation of this graph is an acyclic complete graph, therefore it has the only Hamiltonian path which consists of strongly connected components of initial graph $A_1 \to A_2 \to \dots \to A_k$.
4. The players from the first component A_1 are placed on the first $|A_1|$ places, the players from the component A_2 are placed on the next $|A_2|$ places, and so on.
5. To determine exact place of each player in a strongly connected component, all the procedures from 1 to 5 are repeated recursively inside each component, i.e. for every $i = 1, 2, \dots, k$ players from the component A_i play games with each other again, and so on;
6. If a component consists of a single player, then he has no more rivals, his place is already determined and the process stops.

The players are enumerated with integers from 1 to n. The enumeration was made using results of a previous tournament. It is known that player i wins player j ($i < j$) with probability p.

You need to help to organize the tournament. Find the expected value of total number of games played by all the players.

It can be shown that the answer can be represented as $\frac{P}{Q}$, where P and Q are coprime integers and $Q \not\equiv 0 \pmod{998244353}$. Print the value of $P \cdot Q^{-1}$ modulo 998244353.

If you are not familiar with any of the terms above, you can read about them here.

Input

The first line of input contains a single integer n ($2 \le n \le 2000$) — the number of players.

The second line contains two integers a and b ($1 \le a < b \le 100$) — the numerator and the denominator of fraction $\frac{a}{b} = p$.

Output

In the only line print the expected value of total number of games played by all the players. Print the answer using the format above.

Examples

input

```
3
1 2
```

output

```
4
```

input

```
3
4 6
```

output

```
142606340
```

input

```
4
1 2
```

output

```
598946623
```

Note

In the first example the expected value is 4.

In the second example the expected value is $\frac{27}{7}$.

In the third example the expected value is $\frac{56}{5}$.

G. Power Substring

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given n positive integers $a_1, a_2, \dots, a_n$.

For every a_i you need to find a positive integer k_i such that the decimal notation of 2^{k_i} contains the decimal notation of a_i as a substring among its last $min(100, length(2^{k_i}))$ digits. Here $length(m)$ is the length of the decimal notation of m.

Note that you don't have to minimize k_i. The decimal notations in this problem do not contain leading zeros.

Input

The first line contains a single integer n $(1 \le n \le 2\,000)$ — the number of integers a_i.

Each of the next n lines contains a positive integer a_i $(1 \le a_i < 10^{11})$.

Output

Print n lines. The i-th of them should contain a positive integer k_i such that the last $min(100, length(2^{k_i}))$ digits of 2^{k_i} contain the decimal notation of a_i as a substring. Integers k_i must satisfy $1 \le k_i \le 10^{50}$.

It can be shown that the answer always exists under the given constraints. If there are multiple answers, print any of them.

Examples

input
```
2
8
2
```
output
```
3
1
```

input
```
2
3
4857
```
output
```
5
20
```

H. Don't Exceed

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You generate real numbers $s_1, s_2, \ldots, s_n$ as follows:

- $s_0 = 0$;
- $s_i = s_{i-1} + t_i$, where t_i is a real number chosen independently uniformly at random between 0 and 1, inclusive.

You are given real numbers $x_1, x_2, \ldots, x_n$. You are interested in the probability that $s_i \le x_i$ is true for all i simultaneously.

It can be shown that this can be represented as $\frac{P}{Q}$, where P and Q are coprime integers, and $Q \not\equiv 0 \pmod{998244353}$. Print the value of $P \cdot Q^{-1}$ modulo 998244353.

Input

The first line contains integer n ($1 \le n \le 30$).

The next n lines contain real numbers $x_1, x_2, \ldots, x_n$, given with at most six digits after the decimal point ($0 < x_i \le n$).

Output

Print a single integer, the answer to the problem.

Examples

input

```
4
1.00
2
3.000000
4.0
```

output

```
1
```

input

```
1
0.50216
```

output

```
342677322
```

input

```
2
0.5
1.0
```

output

```
623902721
```

input

```
6
0.77
1.234567
2.1
1.890
2.9999
3.77
```

output

```
859831967
```

Note

In the first example, the sought probability is 1 since the sum of i real numbers which don't exceed 1 doesn't exceed i.

In the second example, the probability is x_1 itself.

In the third example, the sought probability is $3/8$.