## Problem A. Coincidence

| | |
|---|---|
| Input file: | maxcon.im |
| Output file: | maxcom.out |
| Time limit: | 1 second |
| Memory limit: | 64 megabytes |

Common subsequence of two strings $s_1$ and $s_2$ is a pair of sequences of indices $(\{a_i\}, \{b_i\})$ such that $a_1 < a_2 < \ldots < a_k$, $b_1 < b_2 < \ldots < b_k$, and $s_1[a_i] = s_2[b_i]$ for all $1 \le i \le k$.

Find a longest common subsequence of two strings.

### Input

First and second line of an input file contain two strings of French lowercase characters a...z. There are no spaces before, inside or after the strings. Lengths of strings do not exceed 100.

### Output

In the first line of output file output $k$ – the length of a longest common subsequence. On the second line output $k$ numbers – indices of a common subsequence in the first input string. On the third line output the same for the second input string. Index of the first character in the string is 1. Indices should be output in ascending order.

### Example

| maxcon.im | maxcom.out |
|---|---|
| abcd<br>cxbydz | 2<br>3 4<br>1 5 |

## Problem B. Longest Common Subpair

| | |
|---|---|
| Input file: | subpair.in |
| Output file: | subpair.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

A pair of strings $(\alpha, \beta)$ is called a *subpair* of a string $\gamma$ if $\gamma = \gamma_1\alpha\gamma_2\beta\gamma_3$ for some (possibly empty) strings $\gamma_1$, $\gamma_2$ and $\gamma_3$. The length of the pair is the sum of lengths of its strings: $|(\alpha, \beta)| = |\alpha| + |\beta|$.

Given two strings $\xi$ and $\eta$ find their longest common subpair, that is — such pair $(\alpha, \beta)$ that it is a subpair of both $\xi$ and $\eta$ and its length is greatest possible.

### Input

Input file contains two strings $\xi$ and $\eta$, one on a line. Both strings contain only small letters of the English alphabet. Both string are not empty. The length of each string doesn't exceed 3000.

### Output

Output $\alpha$ on the first line of the output file and $\beta$ on the second line.

### Example

| subpair.in | subpair.out |
|---|---|
| abacabadabacaba<br>acabacadacabaca | acaba<br>abaca |
| ab<br>bc | b |

## Problem C. Little Brackets

| | |
|---|---|
| Input file: | brackets.in |
| Output file: | brackets.out |
| Time limit: | 2 seconds |
| Memory limit: | C |

onsider all regular bracket sequences with one type of brackets. Let us call the *depth* of the sequence the maximal difference between the number of opening and the number of closing brackets in a sequence prefix. For example, the depth of the sequence "()(()())" is 2, and the depth of "(((()(()())))" is 4.

Find out the number of regular bracket sequences with $n$ opening brackets that have the depth equal to $k$. For example, for $n = 3$ and $k = 2$ there are three such sequences: "()(())", "(()())", "(())()".

### Input

Input file contains several test cases. Each test case is described with $n$ and $k$ ($1 \le k \le n \le 50$).

Last testcase is followed by two zeroes. They should not be processed.

### Output

For each testcase output the number of regular bracket sequences with $n$ opening brackets that have the depth equal to $k$.

Separate output for different testcases by a blank line. Adhere to the format of the sample output.

### Example

| brackets.in | brackets.out |
|---|---|
| 3 2<br>37 23<br>0 0 | Case 1: 3<br><br>Case 2:<br>203685956218528 |

## Problem D. Travelling Salesman Returns!

| | |
|---|---|
| Input file: | salesman.in |
| Output file: | salesman.out |
| Time limit: | 10 seconds |
| Memory limit: | 64 megabytes |

Travelling Salesman plans to return to the Alpha Centauri system! All the people wait it! They want new best goods from other systems!

But the Salesman as usual wants to minimize the travel expenses. He selects any starting planet, flies there on the intergalactic spaceship, visits all planets in the system in order which minimizes the total cost, and then flies on the intergalactic spaceship away. Of course he does not want to visit any planet more than once. Your task is to calculate the optimal route for the Salesman. The people can wait no longer!

### Input

The Alpha Centauri system contains $n$ planets. This number is written on the first line of the input file ($1 \le n \le 19$). The next $n$ lines contain $n$ numbers each: $j$-th number of the $i$-th line is the travel cost from $i$-th planet to $j$-th. The numbers are separated by spaces. Numbers $a_{ii}$ should be ignored. All numbers are positive integers which do not exceed $10^8$.

### Output

Output the minimal total cost in the first line. In the second

line output $n$ numbers — the route on which the total cost is minimized.

### Example

| salesman.in | salesman.out |
|---|---|
| 3<br><br>8 1 6<br><br>3 5 7<br><br>4 9 2 | 5<br><br>3 1 2 |

## Problem E. Long Dominoes

| | |
|---|---|
| Input file: | dominoes.in |
| Output file: | dominoes.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

Find the number of ways to tile an $m \times n$ rectangle with long dominoes — $3 \times 1$ rectangles.

Each domino must be completely within the rectangle, dominoes must not overlap (of course, they may touch each other), each point of the rectangle must be covered.

### Input

The input file contains $m$ and $n$ ($1 \le m \le 9$, $1 \le n \le 30$).

### Output

Output the number of ways to tile an $m \times n$ rectangle with long dominoes.

### Example

| dominoes.in | dominoes.out |
|---|---|
| 3 3 | 2 |
| 3 10 | 28 |

## Problem F. Holidays

| | |
|---|---|
| Input file: | holidays.in |
| Output file: | holidays.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

You must know that our country is well known for its strange holidays system. We celebrate anything we can, and often new public holidays are announced, though sometimes some are cancelled.

In year 3141 some archeologists have discovered the document that they consider to be the log of changes in the system of public holidays in our country from year $Y_s$ till year $Y_f$, inclusively. Each record of the document has the form

<date-record>, added <date-holiday>
or
<date-record>, removed <date-holiday>

The record of the first type means that the public holiday on <date-holiday> was announced on day <date-record>, and the record of the second type means that the public holiday on <date-holiday> was cancelled on day <date-record>.

Unfortunately, all dates of records only include day and month, but not the year. Now the archeologists wonder, what maximal number of public holidays could have been there during the years described. Your task in this problem is to answer this question and provide the version of the document with years inserted, that would guarantee this number of the holidays.

You must assume that all records in the document are in the chronological order and that there were no day when two different events took place, that is, all dates of records in the document must be different.

Note that if the holiday was announced on its own day, the year it is announced there is no public holiday on this date. Analogously, if the holiday is cancelled on its own day, there is still the holiday this year (so people do not have to go to work after listening to morning radio programs).

Also recall, that the day of February 29 exists only in leap years, that is, years that are divisible by 4, except those divisible by 100, except those divisible by 400. For example, years 1996, 2000 and 2004 are leap, while 1999 or 2100 are not.

### Input

The first line of the input file contains $Y_s$ and $Y_f$ ($1800 \le Y_s \le Y_f \le 2200$, $Y_f - Y_s \le 100$). Next line contains $n$ — the number of records in the document ($1 \le n \le 100$). Next $n$ lines contain the the document records, one on a line. See sample input for more detailed information.

You must not consider any other holidays except those explicitly specified in the document. You may assume that no holiday is removed before it is announced.

### Output

On the first line of the output file print the maximal number of public holidays for the given period. After that print $n$ lines — the version of the document with years inserted that provides the specified number of holidays. Adhere to the format of the sample output.

If it is impossible to interpret the document in the specified way, print $-1$ on the first and the only line of the output file.

### Example

| holidays.in | holidays.out |
|---|---|
| 1900 1999<br>9<br>January 1, added<br>January 1<br>January 1, added<br>January 7<br>February 29, added<br>February 29<br>November 7, added<br>November 7<br>November 7, removed<br>January 7<br>September 1, added May 1<br>August 21, removed<br>November 7<br>September 1, added<br>June 12<br>September 1, added<br>December 12 | 406<br>January 1 1900, added<br>January 1<br>January 1 1901, added<br>January 7<br>February 29 1904,<br>added February 29<br>November 7 1904, added<br>November 7<br>November 7 1905,<br>removed January 7<br>September 1 1906,<br>added May 1<br>August 21 1907,<br>removed November 7<br>September 1 1907,<br>added June 12<br>September 1 1908,<br>added December 12 |

## Problem G. Fibonacci Subsequence

| | |
|---|---|
| Input file: | `fibsubseq.in` |
| Output file: | `fibsubseq.out` |
| Time limit: | 3 seconds |
| Memory limit: | 64 megabytes |

A sequence of integer numbers $a_1, a_2, \ldots, a_n$ is called a *Fibonacci sequence* if $a_i = a_{i-2} + a_{i-1}$ for all $i = 3, 4, \ldots, n$.

Given a sequence of integer numbers $c_1, c_2, \ldots, c_m$ you have to find its longest Fibonacci subsequence.

### Input

The first line of the input file contains $m$ ($1 \le m \le 3\,000$). Next line contains $m$ integer numbers not exceeding $10^9$ by their absolute value.

### Output

On the first line of the output file print the maximal length of the Fibonacci subsequence of the given sequence. On the second line print the subsequence itself.

### Example

| fibsubseq.in | fibsubseq.out |
|---|---|
| 10<br>1 1 3 -1 2 0 5 -1 -1 8 | 5<br>1 -1 0 -1 -1 |

## Problem H. 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<br>3 10<br>7 3<br>8 2 | 1<br>2 |

## Problem I. Order-Preserving Codes

| | |
|---|---|
| Input file: | `codes.in` |
| Output file: | `codes.out` |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

Binary code is a mapping of characters of some alphabet to the set of finite length bit sequences. For example, standard ASCII code is a fixed length code, where each character is encoded using 8 bits.

Variable length codes are often used to compress texts taking into account the frequencies of occurence of different characters. Characters that occur more often get shorter codes, while characters occuring less often — longer ones.

To ensure unique decoding of variable length codes so called *prefix codes* are usually used. In a prefix code no code sequence is a proper prefix of another sequence. Prefix code can be easily decoded scanning the encoded sequence from left to right, since no code is the prefix of another, one always knows where the code for the current character ends and the new character starts.

Among prefix codes, the optimal code is known, so called Huffman code. It provides the shortest possible length of the text among all prefix codes that separatly encode each character with an integer number of bits.

However, as many other codes, Huffman code does not preserve character order. That is, Huffman codes for lexicographically ordered characters are not necessarily lexicographicaly ordered.

In this problem you are asked to develop a prefix code that would be optimal for the given text among all order-preserving prefix codes. Code is called order-preserving if for any two characters the code sequence for the character that goes earlier in the alphabet is lexicographically smaller.

Since text itself is not essential for finding the code, only the number of occurences of each character is important, only this data is given.

### Input

The first line of the input file contains $n$ — the number of characters in the alphabet ($2 \le n \le 2000$). The next line contains $n$ integer numbers — the number of occurences of the characters in the text for which the code must be developed (numbers are positive and do not exceed $10^9$). Characters are described in the alphabetical order.

### Output

Output $n$ bit sequences, one on a line — the optimal order-preserving prefix code for the described text.

### Example

| codes.in | codes.out |
|---|---|
| 5<br>1 8 2 3 1 | 00<br>01<br>10<br>110<br>111 |

## Problem J. Prime Sum

| | |
|---|---|
| Input file: | prime.in |
| Output file: | prime.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

Let us consider a representation of a positive integer number $n$ as a sum of one or more integer numbers:

$$n = x_1 + x_2 + \ldots + x_k.$$

Let us call such sum *prime*, if all terms in it are pairwise relatively prime. Recall, that $x$ and $y$ are called relatively prime, if their greatest common divisor is 1.

Given $n$, find the number of ways it can be represented as a prime sum. The ways that differ only by the order of the terms are considered the same. For example, there are six such representation for $n = 5$:

$$5 = 5$$
$$5 = 4 + 1$$
$$5 = 3 + 2$$
$$5 = 3 + 1 + 1$$
$$5 = 2 + 1 + 1 + 1$$
$$5 = 1 + 1 + 1 + 1 + 1$$

### Input

Input file contains one number $n$ ($3 \le n \le 150$).

### Output

Print one number — the number of ways $n$ can be represented as a prime sum.

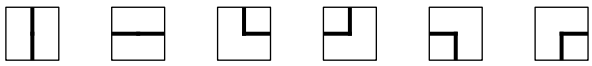
### Example

| prime.in | prime.out |
|---|---|
| 5 | 6 |

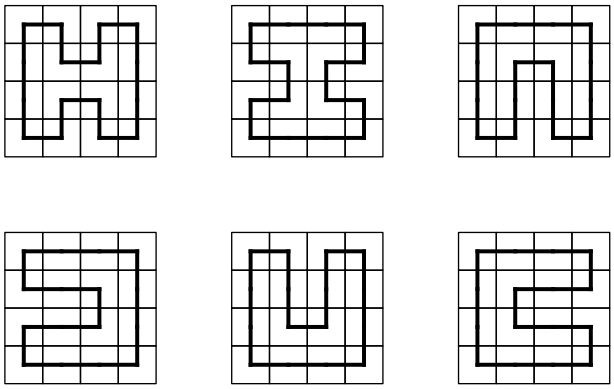
## Problem K. Pipe Layout

| | |
|---|---|
| Input file: | pipe.in |
| Output file: | pipe.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

The city is building a centralized heating system in some of the city districts. City district occupies rectangular area and consists of square blocks arranged in a grid. Centralized heating system is a closed circuit of pipes that will be used to run hot water through every block in the district. The city council is considering different layouts of pipes for each district. In order to minimize the total length of pipes but still provide hot water to every block in the district, exactly one pipe must run through every block. A pipe in every block must be connected to the pipes in two neighboring blocks. So, there are at most six possible pipe configurations in every block:



In order to plan their planning activities, the city council wants to know the number of different possible pipe layouts for a given city district. For example, there are exactly 6 different pipe layouts for a district with 16 blocks arranged into 4 by 4 grid:



### Input

The input file contains two integer numbers $r$ ($r > 1$) and $c$ ($c > 1$). They specify, correspondingly, the number of rows and columns of blocks in the district. The total number of blocks in a district does not exceed 100 ($r \times c \le 100$).

### Output

Output the number of different possible pipe layouts for this district.

### Examples

| pipe.in | pipe.out |
|---|---|
| 4 4 | 6 |
| 5 7 | 0 |
| 2 8 | 1 |
| 12 8 | 102283239429 |

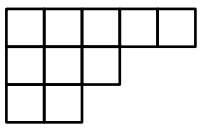
## Problem L. Die Young

| | |
|---|---|
| Input file: | young.in |
| Output file: | young.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

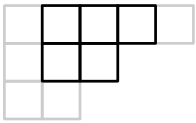
Young diagram is a well known way to describe a partition of a positive integer number. A partition of a number $n$ is a representation as a sum of one or several integer numbers $n = m_1 + m_2 + \ldots + m_k$ where $m_1 \ge m_2 \ge \ldots \ge m_k$.

A diagram consists of $n$ boxes arranged in $k$ rows, where $k$ is the number of terms in the partition. A row representing the number $m_i$ contains $m_i$ boxes. All rows are left-aligned, and sorted from longest to shortest.

The diagram on the picture below corresponds to the partition $10 = 5 + 3 + 2$.



Sometimes it is possible to *inscribe* one Young diagram into the other. Diagram $X$ can be inscribed into the diagram $Y$ if it is possible to delete some boxes from diagram $Y$ so that it turns to diagram $X$. Note that it is only allowed to remove some boxes, it is not allowed to rotate or flip the diagram. For example, the picture below shows that the diagram for $5 = 3+2$ can be inscribed into the diagram for $10 = 5 + 3 + 2$.



On the other hand, for example, it is impossible to inscribe the diagram for $8 = 4 + 4$ into the diagram for $10 = 5 + 3 + 2$.

Given $n$, your task to find such partition of $n$ that the corresponding Young diagram has the greatest possible number of diagrams that can be inscribed into it.

For example, there are 36 Young diagrams that can be inscribed into the diagram for $10 = 5+3+2$. However, it is not the maximal possible value. The diagram for $10 = 4 + 2 + 2 + 1 + 1$ has the better value, there are 41 diagrams that can be inscribed into it.

## Input

Input file contains $n$ ($1 \le n \le 100$).

## Output

At the first line of the output file print the maximal number of Young diagrams that can be inscibed into some Young diagram for the partition of $n$.

At the second line print one or more integer numbers — the number of boxes in each row of the optimal diagram.

## Examples

| young.in | young.out |
|---|---|
| 10 | 41<br>4 2 2 1 1 |

# Problem M. Finite Automata

| | |
|---|---|
| Input file: | automata.in |
| Output file: | automata.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

The *Broken Tiles* company has designed a robot for tiling the roads. The robot has an infinite supply of pavement tiles, each of which has the size of $2 \times 1$ feet. The roads to tile have a size of $m \times n$ feet. Here $m$ is the width of the road, and $n$ is the length of the road.

The program for the robot is the sequence of the commands. There are three commands: '`H`', '`V`' and '`S`'. The robot considers the road to tile consisting of $m \times n$ unit squares. The road runs from west to east.

The robot begins the execution of the program standing in the north-western square of the road. Each step it considers the current command. If it is '`H`', the robot puts the tile horizontally — the longer side from east to west, the western square of the tile occupies the current square. If the command is '`V`', the robot puts the tile vertically — the longer side from north to south, the northern square of the tile occupies the current square. Finally, if the command is '`S`', the robot does nothing on the current square.

After executing the command, the robot moves one square southwards. If it crosses the border of the road, it moves one square eastwards, and moves to the northern square of the new column.

The road is said to be tiled correctly, if all of its squares are covered by the tiles, and no tiles overlap each other or cross the border of the road. The program is said to be correct for $m$, if it causes the robot to tile the $m \times n$ road correctly for some $n$, and after finishing the execution of the program the robot steps out of the southeastern corner of the road. For example, the program "`HHSS`" is correct for 2 (it correctly tiles the road for $n = 2$). The program "`HHV`", in turn, is incorrect for 2 for two reasons: first two tiles overlap with the third tile, and the robot does not leave the road in the end of the program for any $n$.

The designers of the company asked the main programmer of the company to write the program that would verify the correctness of the program for the robot.

But the main programmer of the company has recently learned the theory of finite automata and decided that everything should be programmed using finite automata only. Fortunately, it turned out, that for each $m$ there indeed exists a finite automaton that accepts those and only those strings that form a correct for $m$ program for the robot.

But the main programmer has gone to the Open Automata Documentation summit, leaving you alone with his ideas. The designers (and, more important, managers) are waiting for the verification automaton. So given $m$, you have to create a finite automaton for recognizing the correct programs. Since this is a business project, the automaton must be deterministic.

Recall, that the deterministic finite automaton (DFA) is an ordered set $\langle \Sigma, U, s, T, \varphi \rangle$ where $\Sigma$ is the finite set called *input alphabet* ($\Sigma = \{$`H`, `V`, `S`$\}$ in our case), $U$ is the finite set of *states*, $s \in U$ is the *initial state*, $T \subset U$ is the set of *terminal states* and $\varphi : U \times \Sigma \to U$ is the *transition function*.

The input of the automaton is the string $\alpha$ over $\Sigma$. Initially the automaton is in state $s$. Each step it reads the first character $c$ of the input string and changes its state to $\varphi(u, c)$ where $u$ is the current state. After that the first character of the input string is removed and the step repeats. If the automaton is in the terminal state after its input string is empty, it accepts the initial string $\alpha$, in the other case it rejects it.

## Input

The input file contains $m$ ($1 \le m \le 10$).

## Output

The first line of the output file must contain $u$ — the number of states of the automaton, and $s$ — the initial state (states are numbered from 1 to $u$). The number of states must not exceed 20 000.

The second line must contain $t$ — the number of terminal states, followed by $t$ integer numbers — the terminal states themselves.

The following $u$ lines must contain three numbers each — the $i$-th of these lines must contain $\varphi(i, H)$, $\varphi(i, V)$, and $\varphi(i, S)$.

### Example

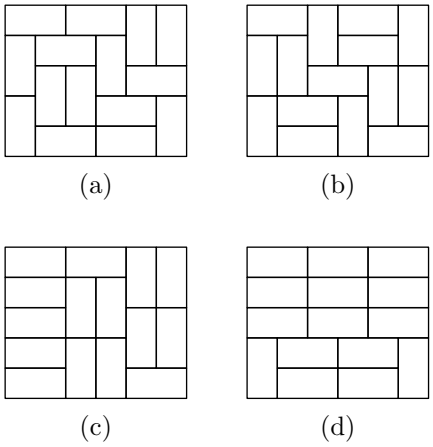
| automata.in | automata.out |
|---|---|
| 2 | 5 1<br>1 1<br>2 3 4<br>5 4 4<br>4 4 1<br>4 4 4<br>4 4 3 |

## Problem N. Solid Tilings

| | |
|---|---|
| Input file: | solid.in |
| Output file: | solid.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

The *Broken Tiles* company's new offer promises its rich clients to pave their rectangular yards with nice $2 \times 1$ and $1 \times 2$ pavement tiles.

The tiling is called *solid* if it is not possible to split the tiled rectangle by a straight line, not crossing the interior of any tile. For example, on the picture below the tilings (a) and (b) are solid, while the tilings (c) and (d) are not.




Now the managers of the company wonder, how many different solid tilings exist for an $m \times n$ rectangle. Help them to find that out.

### Input

The input file contains $m$ and $n$ ($1 \le m \le 8$, $1 \le n \le 16$).

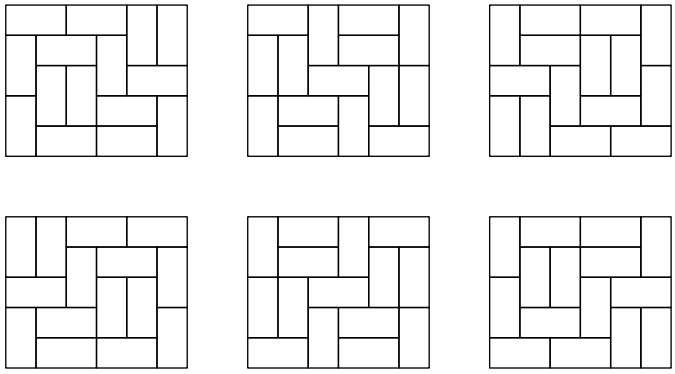
### Output

Output one integer number — the number of solid tilings of $m \times n$ rectangle with $2 \times 1$ and $1 \times 2$ pavement tiles.

### Example

| solid.in | solid.out |
|---|---|
| 2 2 | 0 |
| 5 6 | 6 |

All solid tilings for the $5 \times 6$ rectangle are provided on the picture below:



## Problem O. Paragraph Formatting

| | |
|---|---|
| Input file: | formatting.in |
| Output file: | formatting.out |
| Time limit: | 4 seconds |
| Memory limit: | 64 megabytes |

Mocrosoft company is developing the new publishing software. You are responsible for the module that will do a paragraph layout. You must split the paragraph into lines so that it looked nice.

The paragraph layout problem is well studied, and there are many models that define the value to optimize. One of the most ugly artefacts of the text is the "road" of whitespaces that occurs if three or more whitespaces are one above the other. So the customers asked that the software must avoid such situations as much as possible. This is the problem you must now solve.

You are given one paragraph and must split it into lines to minimize the penalty. The paragraph consists of words. The words are separated by spaces and line feeds. You are not allowed to break words and use hyphenation. The penalty is defined as the sum of lengths of whitespace intervals, such that there is a whitespace immediately above it, and whitespace immediately below it.

All lines except the last one must contain at least two words. The sum of the lengths of the words must not exceed $p - c + 1$ where $p$ is text width, and $c$ is the number of words. The first word of the line is aligned to the left, the last word — to the right. All other words are located in such a way, that all whitespaces have equal width. Since this is the modern software, whitespaces may have a fractional length. However, we consider all characters to have the same width equal to one.

The last line may contain one or more words. Again, the sum of the lengths of the words must not exceed $p - c + 1$ where $p$ is text width, and $c$ is the number of words. The first word of the line is aligned to the left. All other words are located in such a way, that all whitespaces have the width equal to one. The whitespace in the end of the last line is not considered when calculating the penalty.

### Input

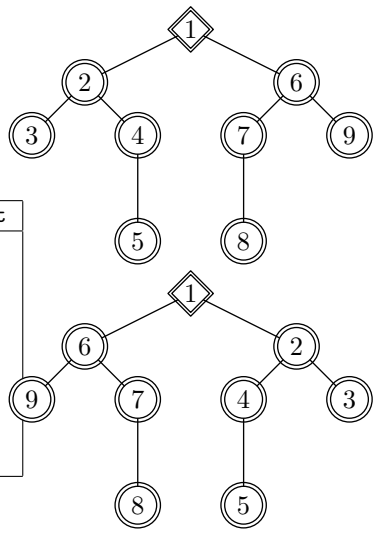
The first line of the input file contains $p$ — the text width ($6 \le p \le 80$). The rest of the file contains the paragraph text — a sequence of words separated by spaces and line feeds. All words consist of 1 to $\lfloor (p-1)/2 \rfloor$ characters with ASCII codes from 33 to 255. The total length of the text does not exceed 5000 characters. The number of words does not exceed 100. The number of short words does not exceed 50 (the word is called short if it contains at most three letters).

## Output

Output one real number — the minimal penalty of the layed out text. You answer must be accurate up to $10^{-4}$.

## Example

| formatting.in | formatting.out |
|---|---|
| 28<br>From thousands of teams taking part in regional contests all around the world seventy eight teams will<br>advance to the 2006 ACM International Collegiate Programming<br>Contest World Finals that will take place in April, in San-Antonio, Texas. | 1.75 |



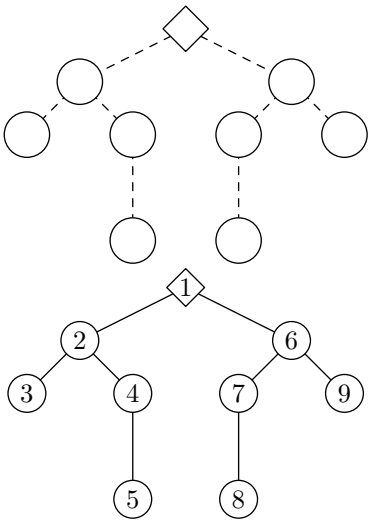

# Problem P. Do It Yourself

| | |
|---|---|
| Input file: | doityourself.in |
| Output file: | doityourself.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

The *Neverplay* company is preparing a new *Do It Yourself* constructor set for children from elementary school. The main part of the constructor is the electronic device that has a quite complicated wires connection scheme.

The wires that are used in the connection scheme have a form of a rooted tree with connection plugs at nodes. The board containing the connection points that must be connected by the wires, has a picture of the tree on it, so it seems quite easy to lay down the wires. Also, the root node of the wires tree has a special form, so the player can always distinguish it from all other nodes.

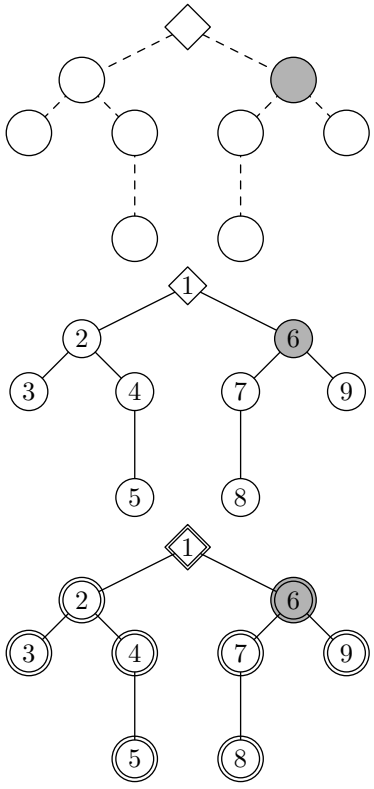
However, all nodes except the root look quite similar, and the information on board is not always enough to connect everything correctly. For example, let us consider the tree shown on the picture below. The picture on the left shows the board. The picture on the right shows the wires tree. Its nodes are numbered for explicitness.




There are two ways the tree can be used to connect the connection points on the board, they are shown on the following picture.

The *Neverplay* developers do not want such ambiguities to exist. They could, of course, mark all connection points on the board and the corresponding plugs of the tree with the same characters. But they want the player to have some challenge when connecting the scheme. So they decided to color the nodes of the tree and the connection points on the boards with several colors. When connecting the scheme the plug at a node of the tree must be inserted into a connection point on the board that has the same color. There must be a unique way to do so.

Now the developers wonder what is the minimal number of colors needed to resolve all ambiguities. The scheme shown on the picture above needs two colors, as shown on the following picture.




Given the wires tree, find out how many colors are needed to mark its nodes, as well as the corresponding connection points on the board, so that there was a unique way to connect the scheme.

### Input

The first line of the input file contains $n$ — the number of nodes of the tree ($1 \le n \le 500$). Let the nodes be numbered from 1 to $n$, so that the number of each node's parent is less than its own number. The root of the tree has number 1.

The second line of the input file contains $n-1$ numbers, for each node from second to $n$-th it contains the number of its parent.

### Output

The first line of the output file must contain a number $k$ — the minimal number of colors needed, so that the scheme could be uniquely connected. The next line must contain $n$ integer numbers from 1 to $k$ — the colors of the vertices.

### Example

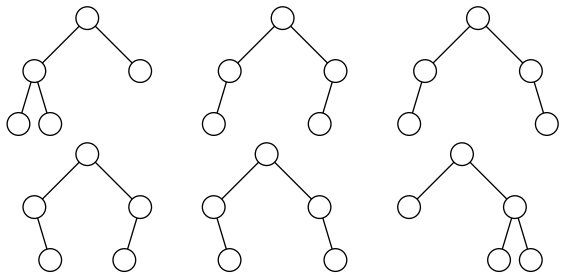
| doityourself.in | doityourself.out |
|---|---|
| 9<br>1 2 2 4 1 6 7 6 | 2<br>1 1 1 1 1 2 1 1 1 |

## Problem Q. AVL Trees

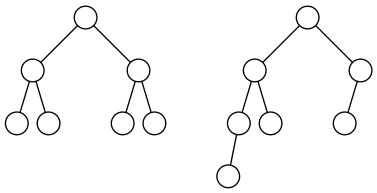
| | |
|---|---|
| Input file: | avl.in |
| Output file: | avl.out |
| Time limit: | 5 seconds |
| Memory limit: | 256 megabytes |

AVL trees invented by Russian scientists Adelson-Velskiy and Landis are used for *sorted collection* data structure. The rooted binary tree is called *balanced* if for each vertex the height of its left subtree and the height of its right subtree differ by at most one. The balanced binary search tree is called the AVL tree.

There can be several AVL trees with the given number of vertices. For example, there are 6 AVL trees with 5 vertices, they are shown on the picture below.



Also the tree with the given number of vertices can have different height, the picture below shows AVL trees with 7 vertices that have height 2 and 3, respectively.



Given $n$ and $h$ find the number of AVL trees that have $n$ vertices and height $h$. Since the answer can be quite large, return the answer modulo 786 433.

### Input

Input file contains $n$ and $h$ ($1 \le n \le 65\,535$, $0 \le h \le 15$)

### Output

Output one number — the number of AVL trees with $n$ vertices that have height $h$, modulo 786 433.

### Example

| avl.in | avl.out |
|---|---|
| 7 3 | 16 |

Note that 786 433 is prime, and $786\,433 = 3 \cdot 2^{18} + 1$.

## Problem R. 2-3 Trees

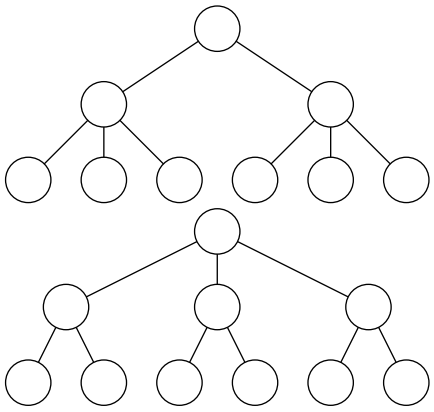
| | |
|---|---|
| Input file: | twothree.in |
| Output file: | twothree.out |
| Time limit: | 2 seconds |
| Memory limit: | 64 megabytes |

2-3 tree is an elegant data structure invented by John Hopcroft. It is designed to implement the same functionality as the binary search tree. 2-3 tree is an ordered rooted tree with the following properties:

- the root and each internal vertex have either 2 or 3 children;
- the distance from the root to any leaf of the tree is the same.

The only exception is the tree that contains exactly one vertex — in this case the root of the tree is the only vertex, and it is simultaneously a leaf, i.e. has no children. The main idea of the described properties is that the tree with $l$ leaves has the height $O(\log l)$.

Given the number of leaves $l$ there can be several valid 2-3 trees that have $l$ leaves. For example, the picture below shows the two possible 2-3 trees with exactly 6 leaves.



Given $l$ find the number of different 2-3 trees that have $l$ leaves. Since this number can be quite large, output it modulo $r$.

### Input

Input file contains two integer numbers: $l$ and $r$ ($1 \le l \le 5\,000$, $1 \le r \le 10^9$).

### Output

Output one number — the number of different 2-3 trees with exactly $l$ leaves modulo $r$.

### Example

| twothree.in | twothree.out |
|---|---|
| 6 1000000000 | 2 |
| 7 1000000000 | 3 |

## Problem S. Block Edit Distance

| | |
|---|---|
| Input file: | `block.in` |
| Output file: | `block.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Recently TopCoder has run a Marathon Match 18 that was intended to support Wikipedia project. The problem that was suggested to the participants had a subproblem — to find a specially defined edit distance between the two words. In this problem we will consider a simplified version of the MM18 problem, but unlike in the Marathon Match we will require the exact solution.

You are given two words $S$ and $T$. First you are allowed to choose some non-intersecting subwords of $S$ and remove them. Each subword you remove costs you $b$. Let the resulting word be $Z$.

After that you must find a standard edit distance between $Z$ and $T$. To do it, you must find the instruction sequence that converts $Z$ to $T$. The allowed instructions are "`I`" — insert, "`D`" — delete, and "`C`" — copy.

Consider that you have two pointers, initially the first pointer is at the first character of $Z$ and the second pointer is at the first character of $T$. "`I`" instruction moves the second pointer one character to the right. "`D`" instruction moves the first pointer one character to the right. "`C`" instruction is applicable only when the two pointers are at equal characters, it moves both pointers one character to the right. Each "`I`" instruction costs $i$, each "`D`" instruction costs $d$, each "`C`" instruction costs $c$.

Find the way to transform $S$ to $T$ in the described way with the smallest cost.

### Input

The first line of the input file contains four integer numbers: $b$, $i$, $d$ and $c$ ($0 \le b, i, d, c \le 10\,000$). The second line contains $S$. The third line contains $T$. The length of each of $S$ and $T$ doesn't exceed 3 000.

### Output

The first line of the output file must contain $K$ — the cost of converting $S$ to $T$ in the described way. The second line must contain $n$ — the number of subwords of $S$ to be removed to form $Z$. The following $n$ lines must contain two integer numbers each — the inclusive ranges of subwords in $S$ to be removed. Characters are numbered from 1.

The last line must contain the sequence of instructions to convert $Z$ to $T$.

### Example

| block.in | block.out |
|---|---|
| 3 1 1 0<br>ABCDEFGHIJKLMN<br>BCDEFZZZZKLM | 9<br>1<br>7 10<br>DCCCCCIIIICCCD |

## Problem T. Discount

| | |
|---|---|
| Input file: | `discount.in` |
| Output file: | `discount.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

The Megga Mall is running a special discount program for its customers. Each customer who buys something really big in the Mall gets $n$ discounts for his purchase. Each discount is presented to the customer as a gift card. The front side of the contains label "$-a_i$", and the reverse side contains label "$-b_i\%$" ($a_i$ and $b_i$ are integer numbers, they can be different for different cards).

Let the initial price of the purchase be $x$. To decrease the price of the purchase the customer can use the gift cards. The cards are used one after another in some arbitrary order. Each card can be used in one of two possible ways: if the card has labels "$-a_i$ and "$-b_i\%$" it can either decrease the current price of the purchase by $a_i$ (front use), or multiply its current price by $1 - b_i/100$ (reverse use). All operations with the price of the purchase are performed with the real number. If the price becomes negative, the purchase is free for the customer.

Peter is going to buy Ferrari in the Megga Mall. The price of Ferrari is $x$. Help Peter to use his cards optimally, so that the final price of Ferrari is minimal possible.

### Input

The first line of the input file contains two integer numbers: $n$ ($1 \le n \le 50$) and $x$ ($1 \le x \le 10^9$). The following $n$ lines describe gift cards, each line contains two integer numbers — $a_i$ and $b_i$ ($1 \le a_i \le 10\,000$, $1 \le b_i \le 99$).

### Output

Output $n$ lines. Lines must describe the gift cards in order they must be used. Each line must contain the number of the card followed by the word "`front`" if the card must be used with front label ("$-a_i$") or the word "`reverse`", if the card must be used with reverse label ("$-b_i\%$").

The final price calculated with your answer must be accurate within $10^{-9}$ of the optimal price (absolute or relative).

### Example

| discount.in | discount.out |
|---|---|
| 3 1000<br>10 1<br>20 1<br>10 2 | 3 reverse<br>1 front<br>2 front |

After applying the third card, Peter gets 2% discount which is 20, so the price becomes 980. After that he uses first and second cards in any order, to get a discount of another $10 + 20 = 30$, so the price becomes 950.

## Problem U. School of Magic

| | |
|---|---|
| Input file: | `school.in` |
| Output file: | `school.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Harry Nomoretter is studying in School of Magic. Today he is planning to start preparing for his first exam. There are four skills that will be tested on the exam: alchemy, basic magic, chronology of magic and demonology.

Harry's experience in each skill can be characterized by an integer number, ranging from 0 to $a$ for alchemy, from 0 to $b$ for basic magic, from 0 to $c$ for chronology of magic, and from 0 to $d$ for demonology. To pass the exam Harry must have each of his skills maximized, that is, his alchemy skill must be equal to $a$, basic magic skill equal to $b$, chronology of magic skill equal to $c$ and demonology skill equal to $d$. However Harry used not to visit lectures, so now his skills are all equal to 0.

To increase his skills Harry can read textbooks. He's got four textbooks: "Applied Alchemy", "Basics of Basic Magic", "Can

Chronology Be As Easy?" and "Demonology for Dummies". Reading a book for an hour increases his alchemy, basic magic, chronology of magic, or demonology skill, respectively, by one.

However, when reading books Harry abstracts from other topics, so his other skills can also change — he can forget something he has read, so the skill decreases, or he can recall something he discussed with his friends before, so the skill increases. Formally, if Harry reads "Applied Alchemy" for one hour, his alchemy skill increases by one, and each of his basic magic, chronology of magic and demonology skills independently uniformly randomly either increases by one, or doesn't change, or decreases by one, with the exception that the skill never decreases below 0, and never increases above its maximum (should the corresponding skill be 0 or maximal, respectively, the uniform choice is made from two variants). Similarly, "Basics of Basic Magic" concerns Harry's basic magic skill, etc.

Now Harry wonders what is the expected time he needs to reach the maximum in each of the skills if he acts optimally. Help him to find that out.

### Input

Input file contains four integer numbers: $a$, $b$, $c$ and $d$ ($1 \le a, b, c, d \le 4$).

### Output

Output one real number — the expected number of hours Harry needs. Your answer must be accurate up to $10^{-6}$.

### Example

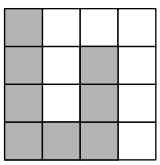
| school.in | school.out |
|---|---|
| 1 1 1 1 | 8.0 |

## Problem V. Separable Divisions

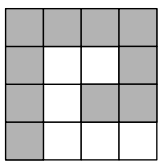
| | |
|---|---|
| Input file: | `separable.in` |
| Output file: | `separable.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider a rectangle that consists of $m \times n$ unit squares. We can divide it into two parts by coloring some squares white and other squares black, so that both white and black parts are connected (the set of squares is called connected if one can walk from any square in the set to any other by stepping from a square to a square that shares an edge with it).

Let us call such division *separable* if one can move white and black parts as far apart from each other as desired, by continuously moving them without overlapping. For example, the division on the figure (a) is separable, but the division on the figure (b) is not.



(a) Separable division.



(b) Nonseparable division.

Find the number of separable divisions of the rectangle.

### Input

Input file contains $m$ and $n$ ($1 \le m, n \le 50$).

### Output

Output one number — the number of separable divisions of an $m \times n$ rectangle.

### Example

| separable.in | separable.out |
|---|---|
| 4 4 | 470 |