Problem A. Alien Microwave

Source file name:	Alien.c, Alien.cpp, Alien.java, Alien.py
Input:	`Standard`
Output:	`Standard`

A standard microwave is operated by entering the cooking time as a string in the form of `hhmmss`, where `hh`, `mm`, and `ss` are two-digit integers less than 24, 60, and 60, respectively. Leading zeros in the string `hhmmss` are omitted. For example, the cooking time of 3 minutes is entered as `300`, though `0300` or `00300` is also accepted.

When any of `hh`, `mm`, or `ss` exceeds the limit, the microwave will not accept it as a valid cooking time and gives an error. For example, `75` is not accepted, nor is `240000`. Note that for the purpose of this problem, we assume that zero seconds of cooking time (represented by a sequence of zero or more 0's) is valid.

Sometimes, one might make a mistake by omitting a digit while entering the cooking time. For example, while entering `1030` (`10` minutes and `30` seconds), omitting the digit `3` turns the input time into `100` (1 minute) instead. Omitting the digit `1` turns it into `030` (30 seconds). In this case, omitting any of the four digits will still make the resulting string a valid cooking time. However, some other strings, while valid cooking times themselves, can become invalid when exactly one of the digits is omitted. For example, `1700` (17 minutes) becomes invalid if either of the zeros is omitted. Such strings are called *Error-Prone* cooking times.

Now, imagine some extraterrestrial planet, on which a standard microwave is operated by a string in the form $a_1a_2a_3 \dots a_n$, where each of a_1, a_2, …, a_n is a two-digit non-negative integer (somehow they also use base 10) less than limits t_1, t_2, …, t_n, respectively. The rules of valid and invalid cooking time still hold.

Given limits t_1, t_2, …, t_n, find the number of *Error-Prone* cooking times. Note that leading zeros don't change the cooking time, so a time specification like `066` is the same as `66`, and should not be counted twice. Also note that `0` is a legitimate cooking time.

Input

The first line of input contains an integer n ($1 \leq n \leq 9$), which is the number of time types in the alien time scheme.

Each of the next n lines contains an integer t_i ($1 \leq t_i \leq 100$), which is the number of partitions in the i^{th} time type in the alien scheme.

Output

Output a single integer, which is the number of *Error-Prone* cooking times without leading zeros.

Example

Input	Output
3 24 60 60	51840

Problem B. Another Substring Query Problem

Source file name:	Another.c, Another.cpp, Another.java, Another.py
Input:	Standard
Output:	Standard

You are given a string s and several queries.

Each query consists of a string t and an integer k. For each query, determine the k^{th} position in s where a substring matching t starts. If t occurs fewer than k times in s, print -1.

Input

The first line of input contains a single string s ($1 \leq |s| \leq 2 \cdot 10^5$), which is the queriable string. It will consist only of lower-case letters.

The next line of input contains a single integer q ($1 \leq q \leq 2 \cdot 10^5$), which is the number of queries that follow.

Each of the next q lines contains a string t ($1 \leq |t|$) and an integer k ($1 \leq k \leq |s|$). This represents a query for the k^{th} occurrence of t in s. The string t will consist only of lower-case letters. The sum of all $|t|$'s will be $\leq 2 \cdot 10^5$

Output

Output a single integer, which is the position of the start of the k^{th} occurrence of t in s, or -1 if t occurs fewer than k times in s. The first character in s is at position 1.

Example

Input	Output
abacabadabacaba 4 a 7 e 3 bac 2 abada 1	13 -1 10 5

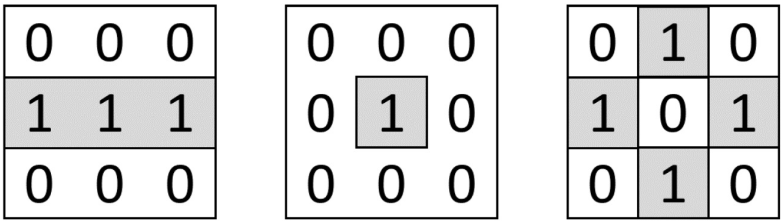
Problem C. Borders

Source file name:	Borders.c, Borders.cpp, Borders.java, Borders.py
Input:	`Standard`
Output:	`Standard`

Consider a black and white image, where every pixel has a value of either 0 or 1. Define a region of the image as a collection of pixels that all have the same value, and are interconnected. Specifically, for any two pixels in a region, there is a path between them only going up, down, left or right, and only going through other pixels with the same value.

You wish to see a border completely around every region in the image. You can choose certain regions to draw a border around; when you do, you draw a border around the entire region, including any internal "holes" (regions entirely contained within the region). If two regions are adjacent, then you can supply the border between then by drawing the border around one, or the other, or both. What is the minimum number of regions you need to draw a border around in order to ensure that every region has a border?

Consider these examples:



- In the first case, the minimum is three. Because they're on the edges, there is no choice but to draw a border around all three.
- In the second case, the minimum is one. Drawing a border around the 0 region puts a border around the 1 region.
- In the third case, the answer is eight. Drawing a border around all of the regions on the edges puts a border around the center region.

Input

The first line of input contains two integers n and m ($1 \le n,\ m \le 100$), where n is the number of rows in the image, and m is the number of columns.

Each of the next n lines contains a single string of length m, consisting only of the characters '0' and '1'. This is the image.

Output

Output a single integer, which is the minimum number of regions you must draw a border around to ensure that every region has a border.

Example

Input	Output

3 3
000
111
000
	3

3 3
000
010
000
	1

3 3
010
101
010
	8

Problem D. Circle of Friends

Source file name:	Circle.c, Circle.cpp, Circle.java, Circle.py
Input:	`Standard`
Output:	`Standard`

There is a posse of friends sitting in a circle. Each friend is holding a card containing a positive integer.

You would like to split the circle of friends into one or more groups. Each group must be a contiguous subsection of the circle. In addition, for each group, the bitwise AND of all values on the cards of the members of the group, taken together, must be nonzero.

Count the number of ways you could split the circle of friends into groups.

Input

The first line of input contains a single integer n ($1 \leq n \leq 2 \cdot 10^5$), which is the number of friends in the circle.

Each of the next n lines contains a single integer a ($1 \leq a < 2^{60}$). These are the positive integers on the cards held by the friends in the circle, in the order that the friends are sitting. Note that since they're in a circle, the last friend in the list is sitting next to the first friend in the list.

Output

Output a single integer, which is the number of ways to split the circle of friends into groups. Since this number may be very large, output it modulo 998.244.353.

Example

Input	Output
4 14 13 11 7	11

Problem E. Finding Polly

Source file name:	Polly.c, Polly.cpp, Polly.java, Polly.py
Input:	`Standard`
Output:	`Standard`

Where is Polly Polygon? She's somewhere in the midst of a field of lines.

Given a set of lines, count how many non-self-intersecting polygons can be formed with exactly one segment from each line. Trivial segments (i.e., length 0) do not count. The polygon must contain exactly the same number of distinct vertices as there are lines in the field.

Input

The first line of input contains an integer n ($3 \leq n \leq 12$), which is the number of lines in the plane.

Each of the next n lines contains four integer coordinate values x_1, y_1, x_2 and y_2 (all between -2000 and 2000 inclusive), representing a line through points $(x_1,\ y_1)$ and $(x_2,\ y2)$. Note that they describe an infinite line, not just a line segment. All lines will be distinct. The two points defining a line will be distinct.

The input lines may be parallel. There may be points where more than two lines intersect. Intersections between lines may occur at points with coordinates greater than 2000.

Output

Output a single integer, which is the number of non-self-intersecting polygons that can be formed with exactly one segment from each line.

Example

Input	Output

4
0 0 0 1
0 0 1 0
0 2 1 0
2 0 0 1
	2

7
0 0 0 1
1 0 1 1
2 0 2 1
0 0 1 0
0 1 1 1
0 2 1 2
0 3 1 3
	0

4
0 1 0 -1
1 0 -1 0
1 1 -1 -1
1 -1 -1 1
	0

Problem F. Investigating Imposters

Source file name:	Imposters.c, Imposters.cpp, Imposters.java, Imposters.py
Input:	`Standard`
Output:	`Standard`

You have stumbled upon a village. In this village, some of the people are "imposters", while the rest are not. Fortunately, you know that the number of possible imposters is limited!

You would like to determine who is not an imposter in this village. To do so, you ask each villager to submit a list of some villagers who are not imposters.

Non-imposters will only submit lists that contain other non-imposter names, while there is no such restriction for imposters. Imposters' lists could contain imposters or non-imposters.

Given the lists of each villager, determine whether they could possibly be an imposter, or are definitely not an imposter.

Input

The first line of input contains two space-separated integers n and k ($1 \leq k \leq n \leq 500$), where n is the number of villagers and k is the maximum possible number of imposters. The villagers are numbered from 1 to n.

Each of the next n lines describes a villager, where the i^{th} line represents the list of villager i. The i^{th} line starts with an integer s ($0 \leq s \leq n$), which is the number of people on villager i's list of non-imposters. Then there will follow s distinct integers denoting the villagers on villager i's list of non-imposters. It is possible for a villager to appear on their own list. All of the villagers on any given list will be distinct.

Output

Output n lines, each with a single integer. The i^{th} line should contain 0 if the villager represented by the i^{th} list in the input could possibly be an imposter, and 1 if that villager is definitely not an imposter.

Example

Input	Output

4 1
1 2
1 1
1 3
1 4
	1
	1
	0
	0

Problem G. Laptop Sticker

Source file name:	Laptop.c, Laptop.cpp, Laptop.java, Laptop.py
Input:	`Standard`
Output:	`Standard`

Your school has provided you with a laptop computer! However, they insist on putting a laptop sticker with their logo on your new computer. That sticker might be very large, and it can't be rotated! Will it fit, with one centimeter to spare on all sides?

Input

The single line of input contains four integers wc, hc, ws and hs ($1 \leq wc,\ hc,\ ws,\ hs \leq 1000$), where wc is the width of your new laptop computer, hc is the height of your new laptop computer, ws is the width of the laptop sticker, and hs is the height of the laptop sticker. All measurements are in centimeters.

Output

Output a single integer, which is 1 if the laptop sticker will fit on your new laptop computer, without rotating, but with one centimeter space on all sides, and 0 if the laptop sticker won't fit.

Example

Input	Output
30 30 27 27	1
35 30 25 29	0
30 35 30 35	0

Problem H. Longest Common Substring

Source file name:	Longest.c, Longest.cpp, Longest.java, Longest.py
Input:	`Standard`
Output:	`Standard`

Given a list of strings, what is the longest substring common to all of them?

Input

The first line of input contains an integer n ($1 \leq n \leq 1000$), which is the number of strings that follow.

Each of the next n lines contains a single string s ($1 \leq |s| \leq 100$) consisting only of lower-case letters.

Output

Output a single integer, which is the length of the longest substring common to all of the input strings.

Example

Input	Output
5 axabcrwmvd abcn tabchwqsl bvrkabcayna tyabc	3

Problem I. Simple Cron Spec

Source file name:	Simple.c, Simple.cpp, Simple.java, Simple.py
Input:	`Standard`
Output:	`Standard`

A *Cron Spec* is used by Linux to specify when to repeatedly execute a certain job. For this problem, consider a *Simple Cron Spec* that defines when a job needs to be run within a single day. A *Simple Cron Spec* has three space separated tokens:

$$hours\ minutes\ seconds$$

that specify at which `hours/minutes/seconds` a job will be run. Hour values are `0-23`, minute and second values are `0-59`.

Each token consists of a single integer value, a value range (two values separated by a dash '`-`'), a comma-separated list of multiple values and/or value ranges, or an asterisk ('`*`'). A value range represents all integer values between the low and high value, inclusive. An asterisk is a special token that represents all possible values. The specified values in any comma separated list must be non-overlapping.

For example, the specification:

```
* 30 20,25,30-33
```

says the job will be run every hour, at minute 30, and second 20, 25, 30, 31, 32, and 33, for a total of $24 \times 1 \times 6 = 144$ times each day.

Given a list of *Simple Cron Specs*, determine two things: First, the number of seconds in the day that at least one job will start, and second, the total number of job starts during the day. Note that if a single job starts 24 times in a day, that counts as 24 job starts.

Input

The first line of input contains a single integer n ($1 \le n \le 100$), which is the number of *Simple Cron Specs* that follow.

Each of the next n lines contains three strings h, m and s. These are the hours, minutes, and seconds specifications of the *Simple Cron Specs*. Each h, m and s consists of either a single star ('`*`'), or a comma-separated list of one or more values or value ranges (two values separated by '`-`'). The values/ranges are guaranteed to be within the appropriate limits (0 to 23 for hours, 0 to 59 for minutes and seconds), and are guaranteed to not overlap. All range specifications are guaranteed to consist of two distinct values, smallest first. All values specified in a comma-separated list are guaranteed to be in strictly increasing order. The whole is guaranteed to be a legitimate *Simple Cron Spec*. The only spaces in the line will be a single space between h and m, and a single space between m and s.

Output

Output two space separated integers. The first is the number of seconds in the 24 hour day with at least one job start, and the second is the total number of job starts.

Example

Input	Output

`2`
`* 30 20,25,30-33`
`9,15 30 *`
	`252 264`

Problem J. Surveillance

Source file name:	Surveillance.c, Surveillance.cpp, Surveillance.java, Surveillance.py
Input:	`Standard`
Output:	`Standard`

You would like to install a single camera somewhere in the interior of a room. The room is described as a polygon with axis-aligned sides. You would like to ensure that the entire area of the room is visible from the camera, modeled as a point that can see in all directions. The complication is that the walls of the room are actually low-quality flat mirrors, such that any part of the room from which light can reach the camera via at most one reflection is visible from the camera (but not two or more reflections). The corners of the room are not reflective.

Given a description of the room, determine the square footage of the area in which the camera may be placed satisfying this condition.

Input

The first line of input contains a single integer n ($4 \leq n \leq 6$), which is the number of corners in the room.

Each of the next n lines contains two integers x and y ($|x|,\ |y| \leq 1000$). These are the room's corners, in counter-clockwise order.

It is guaranteed that the room's walls and corners do not intersect each other, and that all of the walls are axis-aligned. All measurements are in feet.

Output

Output a single floating point number, which is the square footage of the area in which the camera can be placed such that the camera can see the entire area of the room with at most one reflection from the mirrored walls. This value must be accurate to an absolute error of 10^{-6}.

Example

Input	Output

6
0 0
10 0
10 6
4 6
4 10
0 10
	58.666666666666664

Problem K. Train Line

Source file name:	Train.c, Train.cpp, Train.java, Train.py
Input:	`Standard`
Output:	`Standard`

The Intranational Communications Policy Corporation (ICPC) has hired you to complete a new rail line to ensure flow of human and physical capital across a sector of the United States. The path of the railway has already been decided, but where to place the stations is still up for debate. The rail line will be a completely straight line starting at one location, traveling through a number of metropolitan areas, and ending at another location. Each metropolitan area can be represented by a single point on the line with a corresponding population. The ICPC has made the map very detailed in its estimate of the population along the line and now wants to pick the placement of rail stations so as to maximize the utility of the line to the population.

The utility of the line to a particular metropolitan area can be determined as follows. If the area has population p, the value of the line to that population is $p \cdot 2^{-d}$, where d is the distance of the metropolitan area along the line to the nearest rail station. Each metropolitan area only uses the nearest station, so other stations add no utility for that area.

Given the populations of metropolitan areas along the planned route and an upper bound on the maximum number of stations to be placed, determine where to place these stations optimally. Rail stations can be built at any position along the line.

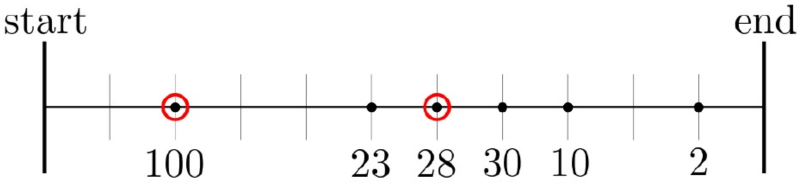


Illustration of example input. Metropolitan areas are depicted as points on the line. The optimal placement of stations is given in red circles, at distances 2.0 and 6.0 from the start of the line, with respective utility 100 and $23 \cdot 1/2 + 28 \cdot 1 + 30 \cdot 1/2 + 10 \cdot 1/4 + 2 \cdot 1/16 = 57.125$.

Input

The first line of input contains two integers n ($1 \leq n \leq 10^5$) and k ($1 \leq k \leq 10^5$), where n is the number of metropolitan areas on the train line, and k is the maximum number of rail stations that may be constructed.

Each of the next n lines contains two integers p ($1 \leq p \leq 100$) and d ($0 \leq d \leq 8 \cdot 10^6$) describing a metropolitan area, where p is the population of the given metropolitan area, and d is the distance of the metropolitan area to the start of the line. No two metropolitan areas will be located at the same distance from the start of the line. For your convenience, the areas will be given in sorted order by distance.

Output

Output a single floating point number, which is the maximum utility possible by constructing at most k rail stations, according to the above utility function. This value must be accurate to an absolute error of 10^{-6}.

Example

Input	Output

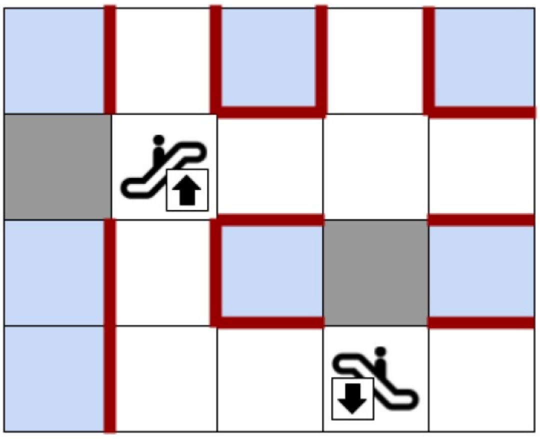
6 2
100 2
23 5
28 6
30 7
10 8
2 10
	157.125

Problem L. Window Shopping

Source file name:	Window.c, Window.cpp, Window.java, Window.py
Input:	`Standard`
Output:	`Standard`

You are to help a newly built shopping mall design its floor layout. The shopping mall can be viewed as a rectangular grid of cells. Two cells are adjacent if they share an edge. A cell may be empty, a pillar, or an escalator. There are exactly two escalators: one goes upstairs and the other downstairs. Note that the escalators are small, so that customers to the mall can pass freely through their locations without going up or down.

You are to choose some of the empty cells to build shops. All remaining empty cells, plus the two escalator cells, are *walkable* by the customers. If an empty cell is reachable from both escalators via some walkable cells, it becomes a *hallway*. A shop window can be installed on an edge between a shop and a hallway.



One optimal way of installing windows for the first sample case. The blue cells are the shops. The red edges are the windows.

After the floor planning, it is allowed to have some empty cells not reachable from both escalators. These empty cells are thus not hallways and cannot have windows installed around them. Also, an escalator is not a hallway and windows cannot be installed on any side of an escalator.

Based on the latest customer analysis report, the profit of the shopping mall grows proportionally to the number of windows inside the mall. You need to therefore determine the maximum number of shop windows that can be installed in the shopping mall.

Input

The first line of input contains two positive integers r and c ($4 \leq r \cdot c \leq 99$), which are the size of the shopping mall. The mall is a grid with r rows and c columns of cells.

Each of the next r lines contains a string of length c. Each character in the string is one of:

a dot '`.`' representing an empty cell

a hash '`#`' representing a pillar

a letter '`U`' representing an Up escalator (Exactly one of these will appear)

a letter '`D`' representing a Down escalator (Exactly one of these will appear)

It is guaranteed that initially all empty cells are reachable from both escalators.

Output

Output a single integer, which is the maximum number of windows the shopping mall can accommodate.

Example

Input	Output
4 5 ..... #U... ...#. ...D.	13
4 4 ..#U ..#D ..#.	5
3 2 ## .D U.	0

Problem M. You Be the Judge!

Source file name:	Yoube.c, Yoube.cpp, Yoube.java, Yoube.py
Input:	`Standard`
Output:	`Standard`

Congratulations! You are now the judge of a programming contest! You've been put in charge of a problem, and since your problem may not have unique correct output, you've got to write an output checker for it.

Your problem is called "Good as Goldbach", and it's based on the Goldbach Conjecture (that any positive even integer greater than 3 can be expressed as the sum of two primes). A solving program's output should have three numbers separated by whitespace: First, a positive even integer greater than 3 and less than or equal to 10^9, and then two (positive) prime numbers which sum to the even number.

You must write a checker for this problem. Your checker should take the output of a contestant's program, and determine whether or not it could possibly be correct. That is, determine if the contestant's output consists only of three tokens separated by whitespace, the first of which is a positive even integer greater than 3 and less than or equal to 10^9, and the next two are positive prime integers which sum to the first. The integers should be in base ten, with no signs or leading zeros. Any amount of white space anywhere except within an integer, including blank lines, is OK. Any other output, extra characters, missing numbers, etc. should be considered incorrect.

Input

The input will consist of from 0 to 1000 lines. Each line consists of from 0 to 100 printable ASCII characters (with codes 32 through 126), or tabs.

Output

Output a single integer, which is 1 if the input could possibly be a correct output for the "Good as Goldbach" problem, or 0 if the input could not possibly be a correct output for the "Good as Goldbach" problem.

Example

Input	Output
10 3 7	1
10 3 7	1

314
159 265
358
	0

22 19 3	1
  60 29 31	1
fred! sam! george!	0