Problem A. A to B

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

You are given two integers, a and b. You want to transform a into b by performing a sequence of operations. You can only perform the following operations:

- Divide a by two (but only if a is even)
- Add one to a

What is the minimum number of operations you need to transform a into b?

Input

The single line of input contains two space-separated integers a and b ($1 \leq a,\ b \leq 10^9$)

Output

Output a single integer, which is the minimum number of the given operations needed to transform a into b.

Example

Input	Output
103 27	4
3 8	5

Problem B. Balanced Animals

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

To save money, Santa Claus has started hiring other animals besides reindeer to pull his sleigh via short term contracts. As a result, the actual animals that show up to pull his sleigh for any given trip can vary greatly in size.

Last week he had 2 buffalo, 37 voles and a schnauzer. Unfortunately, both buffalo were hitched on the left side and the entire sleigh flipped over in mid-flight due to the weight imbalance.

To prevent such accidents in the future, Santa needs to divide the animals for a given trip into two groups such that the sum of the weights of all animals in one group equals the sum of the weights of all animals in the other. To make the hitching process efficient, Santa is seeking an integer target weight t such that all animals that are lighter than t go in one group and those heavier than t go into the other. If there are multiple such t, he wants the smallest one. There's one small wrinkle: what should be done if there some animals have weight exactly equal to t? Santa solves the problem this way: if there are an even number of such animals, he divides them equally among the two groups (thus distributing the weight evenly). But if there are an odd number of such animals, then one of those animals is sent to work with the elves to make toys (it is not put in either group), and the remaining (now an even number) are divided evenly among the two groups.

Input

The first line contains an integer n ($2 \leq n \leq 10^5$), indicating the number of animals.

Each of the next n lines contain a positive integer w ($1 \leq w \leq 2 \cdot 10^5$). These are the weights of the animals (in ounces).

Output

Output the smallest integer target weight t, as described above. It's guaranteed that it is possible to find such an integer.

Example

Input	Output
4 3 6 1 2	4
4 11 8 3 10	10
2 99 99	99

Problem C. Carryless Square Root

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

Carryless addition is the same as normal addition, except any carries are ignored (in base 10). Thus, 37 + 48 is 75, not 85.

Carryless multiplication is performed using the schoolbook algorithm for multiplication, column by column, but the intermediate sums are calculated using *Carryless* addition. Thus:

$$9 \cdot 1234 = 9000 + (900 + 900) + (90 + 90 + 90) + (9 + 9 + 9 + 9)$$

$$= 9000 + 800 + 70 + 6 = 9876$$

$$90 \cdot 1234 = 98760$$

$$99 \cdot 1234 = 98760 + 9876 = 97536$$

Formally, define c_k to be the k^{th} digit of the value c. If $c = a \cdot b$ then

$$c_k = \left[\sum_{i+j=k} a_i \cdot b_j \right] mod\ 10$$

Given an integer n, calculate the smallest positive integer a such that $a \cdot a = n$ in *carryless* multiplication.

Input

The input consists of a single line with an integer n ($1 \leq n \leq 10^{25}$).

Output

Output the smallest positive integer that is a *carryless* square root of the input number, or `-1` if no such number exists.

Example

Input	Output
6	4
149	17
123476544	11112
15	-1

Problem D. Distance

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

The *Levenshtein Distance* between two strings is the smallest number of simple one-letter operations needed to change one string to the other. The operations are:

- Adding a letter anywhere in the string.
- Removing a letter from anywhere in the string.
- Changing any letter in the string to any other letter.

Given a specific alphabet and a particular query string, find all other unique strings from that alphabet that are at a *Levenshtein Distance* of 1 from the given string, and list them in alphabetical order, with no duplicates.

Note that the query string must not be in the list. Its *Levenshtein Distance* from itself is 0, not 1.

Input

Input consists of exactly two lines. The first line of input contains a sequence of unique lower-case letters, in alphabetical order, with no spaces between them. This is the alphabet to use.

The second line contains a string s ($2 \leq |s| \leq 100$), which consists only of lower-case letters from the given alphabet. This is the query string.

Output

Output a list, in alphabetical order, of all strings which are a *Levenshtein Distance* of 1 from the query string s. Output one word per line, with no duplicates.

Example

Input	Output
eg egg	eeg eegg eg ege egeg egge eggg gegg gg ggg

Problem E. Elven Efficiency

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

Like many creatures featured in programming problems, the animals of the forest love playing games with stones. They recently came up with a game to teach the younger animals about divisibility. In this game, each animal starts with a pile of stones. At the start of the game, a series of numbers is called out. For each number that is called, every animal whose number of stones is divisible by the called number scores a point. At the end of the game, the animal with the most points wins.

Emma the forest elf has watched the forest animals play this game many times, and has grown tired of watching the winning animal gloat about how many points they scored. To prevent this from happening, she plans to meddle in the next game the animals play to ensure that no animal scores any points. She plans to wait atop a nearby tree, and keep track of how many stones each animal has. Each round, if an animal is about to score a point, she can toss a stone into that animal's pile, increasing their number of stones by one. The tossed stone stays in that pile for the rest of the game. Throughout the course of the game, she may need to toss several stones into the same pile. But stones are heavy, and she wants to carry as few as possible to the top of her hideout tree. She already knows how many stones each of the n animals will start with, as well as the number to be called out in each of the m rounds of the game, but she wants you to calculate the minimum total number of stones she will have to throw to ensure that no animal scores any points.

Input

The first line of input contains two space-separated integers n and m ($1 \leq n,\ m \leq 10^5$), where n is the number of animals, and m is the number of rounds of the game.

Each of the next n lines contain a single integer a ($1 \leq a \leq 3 \cdot 10^5$), which are the numbers of stones held by each animal.

Each of the next m lines contain a single integer k ($2 \leq k \leq 3 \cdot 10^5$), which are the numbers called out, in order.

Output

Output a single integer, which is the minimum number of stones that Emma must use to prevent any and all animals from scoring any points.

Example

Input	Output
3 5 10 11 12 2 11 4 13 2	12

Problem F. Fixed Point Permutations

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

A permutation of size n is a list of integers $(p_1,\ p_2,\ \ldots,\ p_n)$ from 1 to n such that each number appears exactly once.

The number of *fixed points* of a permutation is the number of indices i such that $p_i = i$.

Given three numbers n, m, and k, find the k^{th} lexicographically smallest permutation of size n that has exactly m *fixed points* (or print `-1` if there are fewer than k permutations that satisfy the condition).

Input

The single line of input contains three space-separated integers n m k ($1 \leq n \leq 50,\ \ 0 \leq m \leq n$, $1 \leq k \leq 10^{18}$), where n is the size of the permutations, m is the number of desired *fixed points*, and the output should be the k^{th} lexicographically smallest permutation of the numbers 1 to n that has exactly m fixed points.

Output

Output the desired permutation on a single line as a sequence of n space-separated integers, or output `-1` if no such permutation exists.

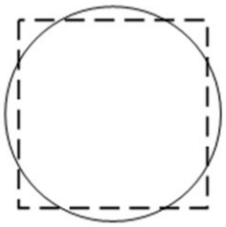
Example

Input	Output
3 1 1	1 3 2
3 2 1	-1
5 3 7	2 1 3 4 5

Problem G. Gauge

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

When going to your internship you got a nice apartment with a skylight. However, one crazy party later and you now have a square-shaped hole where your skylight used to be. Rather than telling the landlord, you decided you would "fix" it by putting a circular pot to collect the water but, as the saying goes, round peg square hole. You need to determine how much of the square the circle covers to help you determine if you should buy a larger pot. Don't worry about the area not covered; you can do multiplication and subtraction easily in your head.



Given the radius of a circular pot and the length of the square skylight, calculate the amount of skylight rain area covered by the pot assuming the two shapes have the same center (i.e., are coaxial) with respect to the direction rain falls from (up). In other words, the center of the square will be directly above the center of the circle. See the picture for an example; let up be the direction from above the page, while the dotted square is the skylight and the solid circle is the pot to collect water.

Input

The first input line contains a positive integer, n, indicating the number of scenarios to check. Each of the following n lines contains a pair of integers, s, r ($1 \leq s \leq 100$, $1 \leq r \leq 100$), which represents the length of the side of the skylight and the radius of the pot, respectively.

Output

For each scenario, output a single decimal representing the area under the skylight that is covered by the pot. Round the answers to two decimal places (e.g., 1.234 rounds to 1.23 and 1.235 rounds to 1.24). For this problem, use 3.14159265358979 as the value of pi.

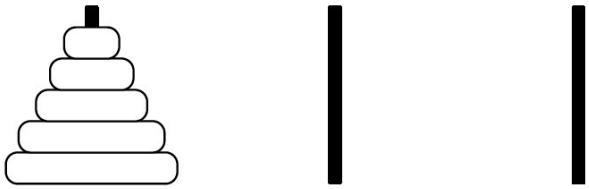
Example

Input	Output
3 1 1 8 5 10 4	1.00 62.19 50.27

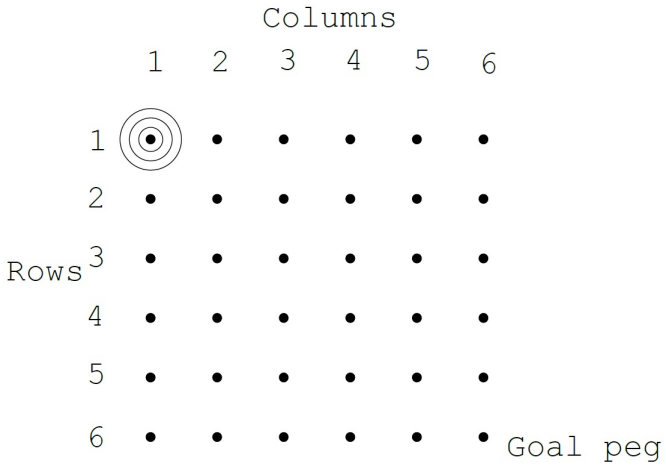
Problem H. Hanoi Grid

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

Towers of Hanoi is a rather famous problem for computer scientists as it is an excellent exercise in recursion. For those of you unfamiliar, here is the classic problem. You are given three pegs. On the first peg, there are d disks placed in decreasing order of size (as placed on the peg). The objective of the game is to move the entire tower from the first peg to the last peg. In each move, you are only allowed to move a single disk from the top of one stack to another stack. For the entire game, no disk of larger size is ever allowed to be placed on top of a disk of smaller size. The goal of the puzzle is to move the tower in the minimum number of moves.



In our problem, we will instead have an $n \times n$ grid of pegs. The rows are numbered top to bottom from 1 to n, while the columns are similarly labeled, from left to right, 1 to n. The original tower is placed on the top left peg $(1,\ 1)$. The goal is to move the tower to the bottom right peg $(n,\ n)$ in the minimum number of moves possible.



Our game will have some different but related rules:

- For a peg $(r,\ c)$ at row r and column c, you may only move the top-most disk from peg $(r,\ c)$ to peg $(r+1,\ c)$ or peg $(c,\ r+1)$, in a single move and only if such a pair of pegs exists.
- Only pegs $(1,\ 1)$ and/or $(n,\ n)$ may have more than one disk at any time; all other pegs may contain at most one disk.
- You can choose any peg for each move.
- You still may never place a larger disk on a smaller disk.

Given the number of disks on the starting peg and the number n described above, determine the minimum number of moves to solve our Tower of Hanoi Grid puzzle.

Input

The first input line contains a positive integer, g, indicating the number of grids to solve. The grids are on the following g input lines, one grid per line. Each grid is described by two integers d and n ($2 \leq d \leq 100$, $2 \leq n \leq 100$), representing the number of disks and the dimensions of the grid, respectively.

Output

For each grid, output "`Grid #d: v`" (without quotes) where v is the minimum number of moves to solve the Tower of Hanoi Grid puzzle. If it is not possible to move the disks from peg $(1,\ 1)$ to peg $(n,\ n)$, output "`impossible`" (without quotes) for v. Leave a blank line after the output for each grid.

Example

Input	Output
3 2 2 100 8 3 100	Grid #1: 4 Grid #2: impossible Grid #3: 594

Problem I. Windmill Pivot

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

Consider a set of points P in the plane such that no 3 points are collinear. Construct a *windmill* as follows:

- Choose a point $p \in P$ and a starting direction such that the line through p in that direction does not intersect any other points in P. Draw that line (Note: *line*, NOT *ray*).
- Rotate the line clockwise like a windmill about the point p as its pivot until the line intersects another point $q \in P$. Designate that point q to be the new pivot, and then continue the rotation. This is called *promoting* point q.
- Continue this process until the line has rotated a full 360°, returning to its original direction (it can be shown that the line will also return to its original position after a 360° rotation).

During this process, a given point in P can be a pivot multiple times. Considering all possible starting pivots and orientations, find the maximum number of times that a single point can be *promoted* during a single 360° rotation of a windmill. Note that the first point is a pivot, but not *promoted* to be a pivot at the start.

Input

The first line of input contains a single integer n ($2 \leq n \leq 2000$), which is the number of points $p \in P$.

Each of the next n lines contains two space-separated integers x and y ($-10^5 \leq x,\ y \leq 10^5$). These are the points. Each point will be unique, and no three points will be collinear.

Output

Output a single integer, which is the maximum number of times any point $p \in P$ can be *promoted*, considering a full 360° rotation and any arbitrary starting point.

Example

Input	Output
3 -1 0 1 0 0 2	2
6 0 0 5 0 0 5 5 5 1 2 4 2	3

Problem J. Jumping Path

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

You are given a rooted tree where each vertex is labeled with a non-negative integer.

Define a *Jumping Path* of vertices to be a sequence of vertices v_1, v_2, ..., v_k where v_i is an ancestor of v_j for all $i < j$. Note that v_i is an ancestor of v_{i+1}, but not necessarily the parent of v_{i+1} (hence the *jumping* part of a *jumping path*).

Compute two quantities:

- The length (number of vertices) of the longest *jumping path* where the labels of the vertices are nondecreasing.
- The number of *jumping paths* of that length where the labels of the vertices are nondecreasing.

Input

The first line of input contains an integer n ($1 \leq n \leq 10^6$), which is the number of vertices in the tree. Vertices are numbered from 1 to n, with vertex 1 being the tree root.

Each of the next n lines contains an integer x ($0 \leq x \leq 10^6$), which are the labels of the vertices, in order.

Each of the next $n-1$ lines contains an integer p ($1 \leq p \leq n$), which are the parents of nodes 2 through n, in order.

It is guaranteed that the vertices form a single tree, i.e., they are connected and acyclic.

Output

Output a single line with two integers separated by a space.

The first integer is length of the longest *jumping path* where the labels of the vertices are nondecreasing. The second integer is the number of *jumping paths* of that length where the labels of the vertices are nondecreasing. As the second integer may be large, give its value modulo 11092019.

Example

Input	Output
5 3 3 3 3 3 1 2 3 4	5 1
5 4 3 2 1 0 1 2 3 4	1 5
4 1 5 3 6 1 2 3	3 2
6 1 2 3 4 5 6 1 1 1 1 1	2 5

Explanation

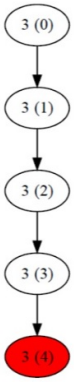


Diagram Example 1.

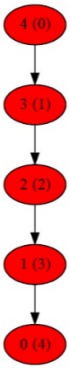


Diagram Example 2.

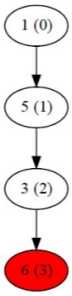


Diagram Example 3.

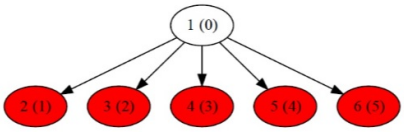


Diagram Example 4.

Problem K. Swap Free

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

A set of words is called *swap free* if there is no way to turn any word in the set into any other word in the set by swapping only a single pair of (not necessarily adjacent) letters.

You are given a set of n words that are all anagrams of each other. There are no duplicate letters in any word. Find the size of the largest *swap free* subset of the given set. Note that it is possible for the largest *swap free* subset of the given set to be the set itself.

Input

The first line of input contains a single integer n ($1 \leq n \leq 500$).

Each of the next n lines contains a single word w ($1 \leq |w| \leq 26$).

Every word contains only lower-case letters and no duplicate letters. All n words are unique, and every word is an anagram of every other word.

Output

Output a single integer, which is the size of the largest *swap free* subset.

Example

Input	Output
6 abc acb cab cba bac bca	3
11 alerts alters artels estral laster ratels salter slater staler stelar talers	8
6 ates east eats etas sate teas	4

Problem L. Lemonade Stand

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

You are running a lemonade stand and have the good fortune of knowing exactly how many cups of lemonade customers are going to want to buy on each day that you run the lemonade stand. You hate to turn any customer away, so you would like to make sure that you always have enough lemons and sugar to make the appropriate number of cups of lemonade on each day. Unfortunately, the cost of lemons and sugar change daily, so you have to choose on which days you buy each, and how much of each to buy. You can buy individual lemons and five pound bags of sugar. (Note that there are 16 ounces in one pound.) On the days you choose to buy ingredients, you buy them in the morning, before any sales are made. (You're an early riser, so you can always get to the store and back before any customers would come.) Note that you can buy as little or as much as you wish on any day to minimize your overall cost, i.e., you have enough startup money (capital) to buy as much as you wish on any day.

Given that you always want to have enough lemons and sugar to serve each customer, determine the minimum cost of buying those lemons and sugar.

Input

The first input line will have a single integer, n ($1 \leq n \leq 100$), the number of cases to process. The first line of each test case will have three space-separated positive integers: d ($1 \leq d \leq 1000$), the number of days you'll run the lemonade stand, x ($1 \leq x \leq 10$), the number of lemons needed to make a single cup of lemonade, and s ($1 \leq s \leq 10$), the number of ounces of sugar needed to make a single cup of lemonade. The following d lines will contain data for days 1 through d, respectively. Each of these lines will have three integers separated by spaces: c ($1 \leq c \leq 1000$), the number of cups sold for that day, p_l ($1 \leq p_l \leq 50$), the price of a single lemon in cents for that day, and p_s ($1 \leq p_s \leq 500$), the price of a five pound bag of sugar in cents for that day. Note that the extra sugar and lemon from each day carry over to the next day.

Output

For each test case, print the minimum cost of supplies (in cents) necessary to make sure that no customer who wants a cup of lemonade gets turned away

Example

Input	Output
2 3 3 2 200 10 399 300 8 499 400 12 499 2 5 10 9 10 199 8 20 99	31977 1347

Problem M. One of Each

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

You are given a sequence of n integers $X = [x_1, x_2, \ldots, x_n]$ and an integer k. It is guaranteed that $1 \leq x_i \leq k$, and every integer from 1 to k appears in the list X at least once.

Find the lexicographically smallest subsequence of X that contains each integer from 1 to k exactly once.

Input

The first line of input contains two integers n and k ($1 \leq k \leq n \leq 2 \cdot 10^5$), where n is the size of the sequence, and the sequence consists only of integers from 1 to k.

Each of the next n lines contains a single integer x_i ($1 \leq x_i \leq k$). These are the values of the sequence X in order. It is guaranteed that every value from 1 to k will appear at least once in the sequence X.

Output

Output a sequence of integers on a single line, separated by spaces. This is the lexicographically smallest subsequence of X that contains every value from 1 to k.

Example

Input	Output
6 3 3 2 1 3 1 3	2 1 3
10 5 5 4 3 2 1 4 1 1 5 5	3 2 1 4 5