Problem A. Ace Arbiter

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

Alice and Bob are playing a friendly game of ping pong. The game consists of several rounds. At the end of each round, exactly one player gets one point. The first player to 11 points wins. When that happens, the game immediately ends. Alice start serving the first round, then Bob serves for the next two rounds, then Alice serves twice, then Bob serves twice, and so on.

At any time in the game, the current score is written X-Y, where X is the number of points of the player who is currently serving, and Y is the number of points of the other player. This constant switching of the scores back and forth can be a little bit error-prone. Eve is the umpire, and she wrote down a log of the current scores at a few different times during the game. But since Eve is a bit unreliable, you worry that she wrote down the wrong scores. Write a program which, given the list of scores Eve wrote down, checks whether or not this list of scores can appear in an actual game.

Input

The first line of input contains an integer n ($1 \leq n \leq 100$), the number of lines in the log. Then follow n lines, each containing a string of the from X-Y ($0 \leq X, Y \leq 11$), the list of scores Eve wrote down, in chronological order.

Output

If the input is a valid log of a not necessarily finished game, output "`ok`". Otherwise, output "error i", where $1 \leq i \leq n$ is the largest value such that the list of the first $i-1$ entries is valid, but the list of the first i entries is invalid.

Example

Input	Output
5 0-0 1-0 1-0 2-0 1-2	ok
2 1-0 0-0	error 2
4 11-0 11-0 11-1 11-11	error 3
4 0-0 1-0 0-2 2-1	error 3

Problem B. Bubble-bubble Sort

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

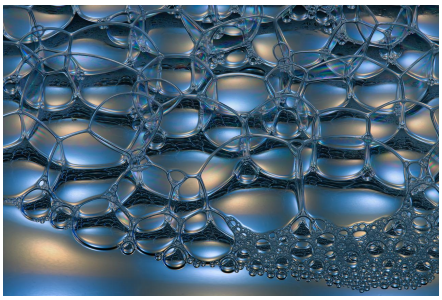


Image by Tom from Pixabay.

Bubbles! As a fanatical supporter of the Bubbles Are Perfect Creatures movement, you have accumulated a large collection of bubbles in all colours and sizes. Being a long time member, your bubbles are among the best in the world, and now is the time to show this. Tomorrow, the yearly Bubble Exposition will be held, and your goal is to win the Bubble Prize and become the Bubble Champion!

However, this is not an easy competition. In order to win, you do not only need the most beautiful bubbles, you also need the best-looking placement of bubbles. You have decided to order the bubbles by bubbliness: less bubblier bubbles to the left, more bubblier bubbles to the right. However, it is hard to compare all the bubbles in your collection at once. In fact, you can only compare up to k bubbles by eye before losing track of all the bubbles. Since your collection consists of more than k bubbles, you need a fancier sorting algorithm.

Your first thought is to use the best sorting algorithm for bubbly purposes, namely Bubble Sort. However, this is the most prestigious bubble competition, so you decide to do better: Bubble-bubble Sort. It works as follows.

Initially, your bubbles are placed in an arbitrary order. Every hour, you do the following: you look at the first k bubbles and place them in the optimal order. Then, you look at bubbles 2 to $k+1$ and place those in the correct order. Then, you look at bubbles 3 to $k+2$, and so on, until you have placed the last k bubbles in the correct order. You then admire how the bubble collection looks so far until the next hour begins and you start at the first bubbles again.

Is this algorithm fast enough to place all your bubbles, or do you need to go further and invent a Bubble-bubble-bubble Sort algorithm? To be precise, after how many hours are the bubbles in the optimal positions?

Input

The input consists of:

- One line with two integers n and k ($2 \leq k < n \leq 2500$), the number of bubbles and the number of bubbles you can sort at once.
- One line with n integers a ($0 \leq a \leq 10^9$), the bubbliness of each bubble in the initial placement of your bubble collection.

Output

Output the number of hours needed to sort your bubble collection.

Example

Input	Output
5 2 3 4 1 5 2	3
8 3 60 8 27 7 68 41 53 44	2
6 3 3 2 4 2 3 1	3

Problem C. Coffee Cup Combo

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

Jonna is a university student who attends n lectures every day. Since most lectures are way too simple for an algorithmic expert such as Jonna, she can only stay awake during a lecture if she is drinking coffee. During a single lecture she needs to drink exactly one cup of coffee to stay awake.

Some of the lecture halls have coffee machines, so Jonna can always make sure to get coffee there. Furthermore, when Jonna leaves a lecture hall, she can bring at most two coffee cups with her to the following lectures (one cup in each hand). But note that she cannot bring more than two coffee cups with her at any given time.

Given which of Jonna's lectures have coffee machines, compute the maximum number of lectures during which Jonna can stay awake.

Input

The first line contains the integers n ($1 \leq n \leq 10^5$), the number of lectures Jonna attends.

The second line contains a string s of length n. The i'th letter is `1` if there is a coffee machine in the i'th lecture hall, and otherwise it is `0`.

Output

Print one integer, the maximum number of lectures during which Jonna can stay awake.

Example

Input	Output
10 0100010100	8
10 1100000000	4
1 0	0

Problem D. Dickensian Dictionary

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

You are stuck in your job at the Boring Accountancy Platform Company; the entire day you have to code all kinds of programs that you do not care about. This involves a lot of tedious typing, which you do not want to do. To pass the time, you decide to interact with the words you type more playfully. In particular, you really enjoy it when you type a word with your left and right hand alternating. You dub these words *Dickensian*.

Flying Fingers. CC BY-NC-ND 2.0
By The Hamster Factor on Flickr

Your mind is quickly overwhelmed with Dickensian words, and even at home they still dictate your thoughts. You want to gather as many Dickensian words as possible and start coding. Given a word, you will need to decide if it is Dickensian or not.

The letters you can type with your left hand are "`qwertasdfgzxcvb`", and the letters you can type with your right hand are "`yuiophjklnm`".

Input

The input consists of:

- One line containing a string of length at least 2 and at most 20, consisting of lowercase characters `a-z`.

Output

Output "`yes`" if the input string is Dickensian, and "`no`" otherwise.

Example

Input	Output
`dickensian`	`yes`
`dictionary`	`no`
`usual`	`yes`
`suspects`	`no`

Problem E. Enigmatic Enumeration

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

Your friend Cajsa had a graph with n vertices, and she needed to find its shortest cycle. To do this, she just took a random sequence of vertices and luckily this happened to be a shortest cycle. "What are the odds?", she asked herself and wrote another program to calculate this probability.

To do this, Cajsa needed an algorithm to count the number of shortest cycles in a graph. She uses a homemade randomized algorithm that runs in $\mathcal{O}(1)$. But you suspect that this algorithm is incorrect, because surely it would have to consider every vertex of the graph (right?). You think that Cajsa's algorithm just prints random numbers that happen to be correct on some small graphs.

In response to these doubts, Cajsa generated a bunch of graphs, and challenges you to check that her answers are correct.

You are given an undirected graph with n vertices and m edges, and your task is to count the number of shortest cycles in it.

A *cycle* in a graph is a path of **distinct** vertices where, additionally, there is an edge between the first and last vertices of the path. Two cycles are considered distinct if they don't consist of the same set of edges. Thus the cycles $3, 1, 2$ and $3, 2, 1$ are the same, and the cycles $1, 2, 3$ and $2, 3, 1$ are considered the same.

Input

The first line of input contains two integers n and m ($3 \leq n \leq 3000$, $3 \leq m \leq 6000$), the number of vertices and the number of edges.

The following m lines each contain two integers u_i and v_i ($1 \leq u_i \neq v_i \leq n$), indicating that an undirected edge goes between u_i and v_i. The graph will not contain duplicate edges or self-loops.

It is guaranteed that the graph contains at least one cycle. However, note that the graph does *not* have to be connected.

Output

Print one integer, the number of shortest cycles in the graph.

Example

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

Problem F. Foreign Football

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

You are on vacation in a foreign country. This country has a local football league, and you don't know any of the team names. However, you have found a table of all the results from this season, and next to every match is the concatenated names of the two teams that played.

There are n teams in total, named $s_1, s_2, \cdots, s_n$. You are given the concatenation $s_i + s_j$ for every ordered pair $i \neq j$. Find the teams names $s_1, s_2, \cdots, s_n$. Team names must be nonempty, but they do not need to be distinct.

Input

The first line of input contains the integer n ($2 \leq n \leq 500$).

The following n lines each contain n strings, the table of concatenated team names. The j:th string on the i:th of these lines will contain the string $s_i + s_j$ if $i \neq j$, and "*" if $i = j$. The concatenated team names will consist of lower case characters a-z.

The total number of characters in concatenated team names is at most 10^6.

Output

If there is no solution, print "`NONE`".

If there is more than one solution, print "`MANY`".

If there is one unique solution, print "`UNIQUE`", followed by n lines containing $s_1, s_2, \cdots, s_n$.

Example

Input	Output
3 * difaik difhammarby aikdif * aikhammarby hammarbydif hammarbyaik *	UNIQUE dif aik hammarby
2 * aaaa aaaa *	MANY
3 * a ab a * b ba b *	NONE
2 * zz zz *	UNIQUE z z

Problem G. Graduation Guarantee

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

Gustav is studying to become an interpreter. In this line of work, knowing several languages is a must, and Gustav is already fluent in French, Mandarin, Nahuatl and even Finnish. But there is one language that Gustav struggles with: Norwegian. Before Gustav can graduate, he must complete the infamous Norwegian Conclusive Proficiency Check.

Picture by Milford, public domain

This exam consists of n yes/no questions. Answering correctly gives $+1$ point, answering incorrectly gives -1 point, and not answering gives 0 points. To pass the exam, at least k points are needed. For each question, Gustav has a guess that he knows is correct with probability p_i ($0.5 \leq p_i \leq 1$). Note that $p_i \geq 0.5$, because otherwise guessing the opposite would be better.

What is the maximum possible probability of passing the exam, assuming we carefully choose which questions to answer and which ones to leave unanswered? Note that unlike a programming contest, the exam does not have instant feedback. So Gustav will answer the questions, hand in his answers, and only later be told the results.

Input

The first line contains the two integers n and k ($1 \leq k \leq n \leq 5000$). The second line contains n real numbers p_i ($0.5 \leq p_i \leq 1.0$). These numbers will have at most 6 digits after the decimal point.

Output

Output the probability that we pass the exam. Your answer will be considered correct if it has an absolute error of at most 10^{-6}.

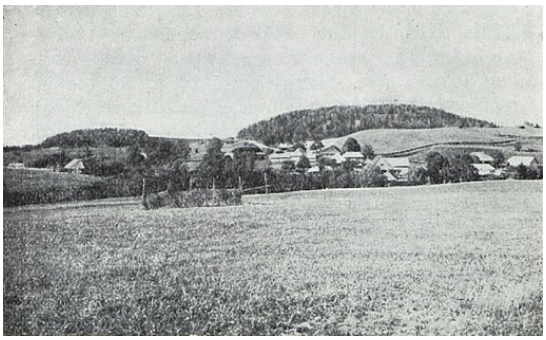
Example

Input	Output
3 3 0.5 0.5 0.5	0.125
4 1 0.9 0.5 0.9 0.9	0.972

Problem H. Highest Hill

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

Sweden may not have a particularly impressive mountain range compared to other NCPC countries such as Norway and Iceland, but at least it beats the flatlands of Denmark. The situation is not so clear when comparing other member countries though. For example, is Estonia more mountainous than Lithuania[1]? To settle this question, you want to determine which of the two countries has the most impressive mountain peak.



An old photograph of Suur Munamägi, public domain

A mountain range is defined by sampling the heights h_i of n equidistant points. Within a mountain range, we call a triple of indices $1 \leq i < j < k \leq n$ a *peak* if $h_i \leq \cdots \leq h_j \geq \cdots \geq h_k$. The *height* of a peak is defined as the smaller of $h_j - h_i$ and $h_j - h_k$.

Given a mountain range, can you find the height of its highest peak?

Input

The first line contains a single integer N ($3 \leq n \leq 200\,000$), the number of sampled points of the mountain range.

The second and final line contains the heights $h_1, \ldots, h_N$ ($0 \leq h_i \leq 318 \cdot 10^9$) of the sampled points, in nanometers above sea level.

It is guaranteed that the mountain range contains at least one peak.

Output

Output a single integer: the height of the highest peak.

Example

Input	Output
11 0 1 2 3 4 5 4 3 2 1 0	5
10 29 85 88 12 52 37 19 86 7 44	67
3 2147483647 318000000000 2147483647	315852516353
3 1 1 1	0

[1]Yes, but not by much! The highest point in Lithuania is Aukštojas Hill, 293.84 meters above sea level, while Estonia has the highest peak in the Baltics: Suur Munamägi is 318 meters above sea level.

Problem I. You be The Judge, Again

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

You are a judge, again! The contest you're judging includes the following problem:

"You have one L-shaped triomino of each of $\frac{4^n-1}{3}$ different colors. Tile a 2^n by 2^n grid using each of these triominos such that there is exactly one blank square and all other squares are covered by exactly one square of such a triomino. All triominos must be used."

Your team is to write a checker for this problem. Validation of the input values and format has already taken place. You will be given a purported tiling of a 2^n by 2^n grid, where each square in the grid is either 0 or a positive integer from 1 to $\frac{4^n-1}{3}$ representing one of the colors. Determine if it is, indeed, a covering of the grid with $\frac{4^n-1}{3}$ unique triominos and a single empty space.

L-shaped triominos look like this:

Input

The first line of input contains a single integer n ($1 \leq n \leq 10$), which is the n of the description.

Each of the next 2^n lines contains 2^n integers x ($0 \leq x \leq \frac{4^n-1}{3}$), where 0 represents an empty space, and any positive number is a unique identifier of a triomino.

Output

Output a single integer, which is 1 if the given grid is covered with $\frac{4^n-1}{3}$ unique triominos and a single empty space. Otherwise, output 0.

Example

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

Problem J. Jack the Mole

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

CC BY-SA 3.0 By Kenneth Catania

The spy agency you are employed by[2] has sent you and some fellow spies on an important mission deep in the jungles of Luxembourg. Your colleagues and you are currently waiting on an abandoned and overgrown airstrip, about to be picked up by the finest spy plane the Dutch bureaucracy can buy.

However, an urgent communication from the HQ has just come in to inform you counterintelligence has learned that during the mission, an infamous counter-spy called Jack the Mole was planted among you! You do a headcount and to your horror realize that while you started the mission with $n-1$ people, there are now $n!$[3] Since you all take Covid regulations very seriously, you all wore masks, so it is not obvious who the imposter is.

Then you remember: the finest spy plane in question is thrown off balance rather easily, so the agency picked the $n-1$ agents so that it is possible to divide them into two groups of equal weight, but not necessarily of equal size, for the left and the right side of the plane respectively. However, it did not specify exactly which two groups (anonymity and all that). Of course you have a scale with you (doesn't any good spy?), so you can find which of the n people on the mission is potentially the mole.

Input

The input consists of:

- One line containing the number of spies present, n ($3 \leq n \leq 300$).
- One line containing n space separated integers $w_1, \ldots, w_n$ ($1 \leq w_i \leq 1000$), the ith of which gives the weight of the ith spy in kilograms[4].

Output

Output a single number k, the number of spies that are potentially the mole, followed by k integers, the suspects, in increasing order.

Example

Input	Output
3 1 1 2	1 3
3 1 1 1	3 1 2 3
5 2 1 3 4 2	3 1 4 5
5 1 1 1 3 1	5 1 2 3 4 5
4 2 1 5 3	2 2 3

[2] When your family members ask, please lie to them and say you work as a computer programmer.
[3] Exclamation point, not factorial.
[4] If those ranges seem odd, recall spies can be very muscular or very nimble.

Problem K. Keyboard Queries

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

Katrín and her friends are university students, and they go to a seminar every week. At the start of each seminar, the professor divides the students into groups randomly. Katrín and her friends dislike random groups, they want to form a group together so they can chat with each other and not get to know the other students. The professor has a secret string S consisting of n characters on their computer. This string acts as a seed for the random group generation. When generating groups, a program `manager` is ran with a substring of S as input. But sometimes, the professor mistypes and writes `manacher` instead. This will indicate that the substring is a palindrome. Maybe Katrín can use this information to predict what the group division will look like?

There is a secret string S with n characters in an unknown alphabet. You are given q queries of two types.

- `1 l r`: This means that the substring of S at indices l through r is a palindrome.
- `2 a b x y`: This means that you should determine whether the substring at indices a though b is equal to the substring at indices x through y, given the information in the previous queries.

Input

The first line of input contains two integers n and q ($1 \leq n \leq 10^5$, $1 \leq q \leq 2 \cdot 10^5$), the number of characters in the unknown string and the number of queries.

The following q lines each start with a 1 or a 2 indicating the type of query. For queries of type 1, two integers l and r follow ($1 \leq l \leq r \leq n$), meaning that the corresponding substring of S is a palindrome. For queries of type 2, four integers a, b, x, y follow ($1 \leq a \leq b \leq n$, $1 \leq x \leq y \leq n$), meaning that you should determine whether those two substrings are equal or not.

Output

For each query of the second kind print "`Equal`" if the substrings must be equal, "`Not equal`" if the substrings can't be equal, and "`Unknown`" if either possibility could be true given the information thus far.

Example

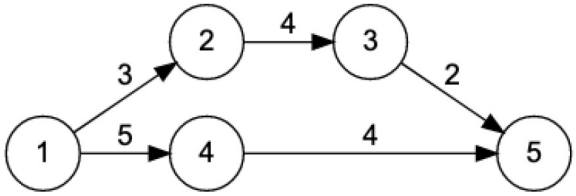
Input	Output
6 8 1 1 6 2 1 1 6 6 2 1 2 5 6 2 1 3 5 6 1 1 3 2 1 3 4 6 2 4 4 6 6 2 2 3 4 5	Equal Unknown Not equal Equal Equal Unknown

Problem L. TraveLog

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

After a long time apart, Alice and Bob have reunited. They live in a country with n cities, creatively named city 1 to city n. Bob drove from his home in city 1 to Alice's home in city n.

When Alice asked Bob which path he took, he was stunned to find he didn't remember. Bob is efficient, and drove without stopping, and knows there is no path faster than the one he took. He also has a brand new National Adventuring Company (NAC) TraveLog! Every time Bob drives through a city, the TraveLog records the time between when he left city 1 and when he arrived in the current city.



In the above layout, there are two possible fastest paths Bob can take from city 1 to city n: $1 \to 2 \to 3 \to 5$ or $1 \to 4 \to 5$. Both paths take a total of 9 units of time. The first path would have a TraveLog of 0, 3, 7, 9, and the second would have a TraveLog of 0, 5, 9.

Unfortunately, the memory in Bob's TraveLog is corrupted. Bob thinks that some of the times are gone, and the remaining times are shuffled arbitrarily. Given what remains of the TraveLog, can you reconstruct Bob's path?

Input

The first line of input contains three integers n ($2 \leq n \leq 2 \cdot 10^5$), m ($1 \leq m \leq 3 \cdot 10^5$) and d ($1 \leq d \leq n$), where n is the number of cities in the country, m is the number of one-way roads between those cities, and d is the number of times remaining in Bob's corrupted TraveLog. The cities are identified by number, 1 through n. Bob lives in city 1, and Alice lives in city n.

Each of the next m lines contains three integers u, v ($1 \leq u,\ v \leq n,\ u \neq v$) and h ($1 \leq h \leq 10^6$). Each line describes a one-way road from city u to city v that takes h units of time to traverse. There is guaranteed to be at least one path from city 1 to city n. There may be several roads between any given pair of cities.

Each of the next d lines contains a single integer t ($0 \leq t \leq 10^{18}$). This is what remains of Bob's TraveLog. Each line is a record of the amount of time Bob took driving from city 1 to another city on his path. These values are guaranteed to be distinct.

Output

The output format depends on the number of paths consistent with Bob's TraveLog.

- If there is no path consistent with Bob's TraveLog, output 0.
- If multiple paths are consistent with Bob's TraveLog, output 1.
- Otherwise, on the first line, output the number of cities on Bob's path. On subsequent lines, output the cities Bob visited, one per line, in the order he visited them.

Example

Input	Output
5 5 2 1 2 3 2 3 4 3 5 2 1 4 5 4 5 4 5 9	3 1 4 5
6 8 2 1 2 1 2 3 2 3 6 8 1 4 3 4 5 4 5 6 4 5 2 7 1 6 13 0 3	1
2 1 1 1 2 10 5	0

Problem M. Mountainous Palindromic Subarray

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

An array is *Mountainous* if it is strictly increasing, then strictly decreasing. Note that *Mountainous* arrays must therefore be of length three or greater.

A *Subarray* is defined as an array that can be attained by deleting some prefix and suffix (possibly empty) from the original array.

An array or subarray is a *Palindrome* if it is the same sequence forwards and backwards.

Given an array of integers, compute the length of the longest *Subarray* that is both *Mountainous* and a *Palindrome*.

Input

The first line of input contains an integer n ($1 \leq n \leq 10^6$), which is the number of integers in the array.

Each of the next n lines contains a single integer x ($1 \leq x \leq 10^9$). These values form the array. They are given in order.

Output

Output a single integer, which is the length of the longest *Mountainous Palindromic Subarray*, or -1 of no such array exists.

Example

Input	Output
8 2 1 2 3 2 1 7 8	5
5 2 5 8 7 2	-1