1. University Schedule
2. 775A

time limit per test

10 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

In this problem your task is to come up with a week schedule of classes in university for professors and student groups. Consider that there are 6 educational days in week and maximum number of classes per educational day is 7 (classes numerated from 1 to 7 for each educational day).

It is known that in university *n* students study, *m* professors work and there are *a* classrooms for conducting classes. Also you have two-dimensional array with *n* × *m* size which contains the following information. The number which stays in *i*-th row and *j*-th column equals to the number of classes which professor *j* must conduct with the group *i* in a single week. The schedule which you output must satisfy to array described above.

There are several other conditions for schedule. Single professor can not conduct more than one class. Similarly, single student group can not be on more than one class at the same time.

Let define a *fatigue* function for professors and student groups. Call this function *f*.

To single professor *fatigue* calculated in the following way. Let look on classes which this professor must conduct in each of the 6-th educational days. Let *x* be the number of class which professor will firstly conduct in day *i* and let *y* — the last class for this professor. Then the value (2 + *y* - *x* + 1)·(2 + *y* - *x* + 1) must be added to professor's *fatigue*. If professor has no classes in day *i*, nothing is added to professor's *fatigue*.

For single student group *fatigue* is calculated similarly. Lets look at classes of this group in each of the 6 educational days. Let *x* be the number of first class for this group on day *i* and let *y* — the last class for this group. Then the value (2 + *y* - *x* + 1)·(2 + *y* - *x* + 1) must be added to this group's *fatigue*. If student group has no classes in day *i*, nothing is added to group's *fatigue*.

So the value of function *f* equals to total {fatigue} for all *n* student groups and for all *m* professors.

Your task is to come up with such a schedule which minimizes the value of function *f*.

Jury prepared some solution of this problem. For each test you will get a certain number of points. It equals to result of division of the value of function *f* from the jury solution by the value of function *f* for schedule which your program output (i. e. the smaller value of {fatigue} function your program find the more points you will get), multiplied by 100. In the other words if the value of *f* for jury solution equals to *p* and for your solution — to *q*, you will get 100·*p* / *q* points (note, that the number of points is a real number). The points will be added together for all tests. The goal is to score as many points as possible.

**Input**

The first line contains three integers *n*, *m* and *a* (1 ≤ *n*, *m*, *a* ≤ 60) — the number of groups, the number of professors and the number of classrooms.

Each of the following *n* lines contains *m* integers from 0 to 24 — *j*-th number in *i*-th line equals to the number of classes with the professor *j* must conduct with the *i*-th student group.

It is guaranteed that the number of classes in week for each professor and for each student group does not exceed 24. Also guaranteed that the total number of classes in week does not exceed 75% from a maximum number of classes which can be conducted based on the number of classrooms. For all tests there is at least one schedule satisfying all described conditions.

**Output**

In the first line print the minimized value of function *f*.

After that print blank line.

After that print the schedule for each student group in increasing order of group number. For each student group print 7 lines. Each line must contains 6 numbers. Let the number at *i*-th line and *j*-th column equals to *x*. If in *j*-th day current group has no class number *i*, *x*must be equals to zero. Otherwise *x* must be equals to the number of professor who will conduct the corresponding class with the corresponding student group.

The number of classes which will be conducted simultaneously must not exceeds the number of classrooms *a*.

Separate the description of the schedules for groups with a blank line.

**Examples**

**input**

3 3 1  
1 0 0  
0 1 0  
0 0 1

**output**

54  
  
1 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
  
0 0 0 0 0 0   
2 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
  
0 0 0 0 0 0   
0 0 0 0 0 0   
3 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0

**input**

3 1 1  
1  
1  
1

**output**

52  
  
1 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
  
0 0 0 0 0 0   
1 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
  
0 0 0 0 0 0   
0 0 0 0 0 0   
1 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0

**input**

5 7 10  
1 3 6 0 1 2 4  
0 3 0 6 5 1 4  
3 5 1 2 3 2 4  
2 3 1 1 4 1 2  
2 4 3 2 4 3 2

**output**

1512  
  
0 0 6 0 0 2   
0 7 6 3 3 7   
3 1 2 3 2 7   
3 7 0 0 0 0   
5 3 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
  
0 0 4 0 7 6   
4 5 7 4 5 5   
7 2 4 4 5 5   
7 2 0 4 0 0   
0 2 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
  
4 0 7 2 5 7   
5 0 2 5 7 1   
2 4 1 2 7 1   
2 3 0 0 0 0   
0 6 0 0 0 0   
0 6 0 0 0 0   
0 0 0 0 0 0   
  
0 0 0 5 3 5   
0 2 4 7 2 6   
0 5 7 0 0 0   
1 5 1 0 0 0   
2 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0   
  
0 0 5 7 2 3   
0 1 3 2 6 3   
5 7 6 5 6 4   
5 4 2 2 0 0   
1 0 0 0 0 0   
0 0 0 0 0 0   
0 0 0 0 0 0

**Note**

During the main part of the competition (one week) you solution will be judged on 100 preliminary tests. The first 10 preliminary tests are available for download by a link <http://assets.codeforces.com/files/vk/vkcup-2017-wr2-materials-v1.tar.gz>.

After the end of the contest (i.e., a week after its start) the last solution you sent (having positive score) will be chosen to be launched on the extended final tests.

L. Bars

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Polycarp's workday lasts exactly *n* minutes. He loves chocolate bars and can eat one bar in one minute. Today Polycarp has *k* bars at the beginning of the workday.

In some minutes of the workday Polycarp has important things to do and in such minutes he is not able to eat a chocolate bar. In other minutes he can either eat or not eat one chocolate bar. It is guaranteed, that in the first and in the last minutes of the workday Polycarp has no important things to do and he will always eat bars in this minutes to gladden himself at the begining and at the end of the workday. Also it is guaranteed, that *k* is strictly greater than 1.

Your task is to determine such an order of eating chocolate bars that the maximum break time between eating bars is as minimum as possible.

Consider that Polycarp eats a bar in the minute *x* and the next bar in the minute *y* (*x* < *y*). Then the break time is equal to *y* - *x* - 1minutes. It is not necessary for Polycarp to eat all bars he has.

**Input**

The first line contains two integers *n* and *k* (2 ≤ *n* ≤ 200 000, 2 ≤ *k* ≤ *n*) — the length of the workday in minutes and the number of chocolate bars, which Polycarp has in the beginning of the workday.

The second line contains the string with length *n* consisting of zeros and ones. If the *i*-th symbol in the string equals to zero, Polycarp has no important things to do in the minute *i* and he can eat a chocolate bar. In the other case, Polycarp is busy in the minute *i* and can not eat a chocolate bar. It is guaranteed, that the first and the last characters of the string are equal to zero, and Polycarp always eats chocolate bars in these minutes.

**Output**

Print the minimum possible break in minutes between eating chocolate bars.

**Examples**

**input**

3 3  
010

**output**

1

**input**

8 3  
01010110

**output**

3

**Note**

In the first example Polycarp can not eat the chocolate bar in the second minute, so the time of the break equals to one minute.

In the second example Polycarp will eat bars in the minutes 1 and 8 anyway, also he needs to eat the chocolate bar in the minute 5, so that the time of the maximum break will be equal to 3 minutes.

K. Stepan and Vowels

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Stepan likes to repeat vowel letters when he writes words. For example, instead of the word "pobeda" he can write "pobeeeedaaaaa".

Sergey does not like such behavior, so he wants to write a program to format the words written by Stepan. This program must combine all consecutive equal vowels to a single vowel. The vowel letters are "a", "e", "i", "o", "u" and "y".

There are exceptions: if letters "e" or "o" repeat in a row exactly 2 times, like in words "feet" and "foot", the program must skip them and do not transform in one vowel. For example, the word "iiiimpleeemeentatiioon" must be converted to the word "implemeentatioon".

Sergey is very busy and asks you to help him and write the required program.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 100 000) — the number of letters in the word written by Stepan.

The second line contains the string *s* which has length that equals to *n* and contains only lowercase English letters — the word written by Stepan.

**Output**

Print the single string — the word written by Stepan converted according to the rules described in the statement.

**Examples**

**input**

13  
pobeeeedaaaaa

**output**

pobeda

**input**

22  
iiiimpleeemeentatiioon

**output**

implemeentatioon

**input**

18  
aeiouyaaeeiioouuyy

**output**

aeiouyaeeioouy

**input**

24  
aaaoooiiiuuuyyyeeeggghhh

**output**

aoiuyeggghhh

J. Stepan's Series

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Well, the series which Stepan watched for a very long time, ended. In total, the series had *n* episodes. For each of them, Stepan remembers either that he definitely has watched it, or that he definitely hasn't watched it, or he is unsure, has he watched this episode or not.

Stepan's dissatisfaction is the **maximum** number of consecutive series that Stepan did not watch.

Your task is to determine according to Stepan's memories if his dissatisfaction could be exactly equal to *k*.

**Input**

The first line contains two integers *n* and *k* (1 ≤ *n* ≤ 100, 0 ≤ *k* ≤ *n*) — the number of episodes in the series and the dissatisfaction which should be checked.

The second line contains the sequence which consists of *n* symbols "Y", "N" and "?". If the *i*-th symbol equals "Y", Stepan remembers that he has watched the episode number *i*. If the *i*-th symbol equals "N", Stepan remembers that he hasn't watched the epizode number *i*. If the *i*-th symbol equals "?", Stepan doesn't exactly remember if he has watched the episode number *i* or not.

**Output**

If Stepan's dissatisfaction can be exactly equal to *k*, then print "YES" (without qoutes). Otherwise print "NO" (without qoutes).

**Examples**

**input**

5 2  
NYNNY

**output**

YES

**input**

6 1  
????NN

**output**

NO

**Note**

In the first test Stepan remembers about all the episodes whether he has watched them or not. His dissatisfaction is 2, because he hasn't watch two episodes in a row — the episode number 3 and the episode number 4. The answer is "YES", because *k* = 2.

In the second test *k* = 1, Stepan's dissatisfaction is greater than or equal to 2 (because he remembers that he hasn't watch at least two episodes in a row — number 5 and number 6), even if he has watched the episodes from the first to the fourth, inclusive.

I. Composing Of String

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Stepan has a set of *n* strings. Also, he has a favorite string *s*.

Stepan wants to do the following. He will take some strings of his set and write them down one after another. It is possible that he will take some strings more than once, and will not take some of them at all.

Your task is to determine the minimum number of strings in the set which Stepan needs to take and write so that the string *s* appears as a subsequence in the resulting written down string.

For example, in the string "abcd" strings "ad", "acd", "abcd" appear as subsequences, and strings "ba", "abdc" don't appear as subsequences.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 50) — the number of strings in Stepan's set.

The next *n* lines contain *n* non-empty strings consisting of lowercase letters of the English alphabet. The length of each of these strings does not exceed 50 symbols. It is possible that some strings from Stepan's set are the same.

The next line contains the non-empty string *s*, consisting of lowercase letters of the English alphabet — Stepan's favorite string. The length of this string doesn't exceed 2500 symbols.

**Output**

Print the minimum number of strings which Stepan should take from the set and write them down one after another so that the string *s*appears as a subsequence in the resulting written down string. Each string from the set should be counted as many times as Stepan takes it from the set.

If the answer doesn't exsist, print -1.

**Examples**

**input**

3  
a  
aa  
a  
aaa

**output**

2

**input**

4  
ab  
aab  
aa  
bb  
baaab

**output**

3

**input**

2  
aaa  
bbb  
aaacbbb

**output**

-1

**Note**

In the first test, Stepan can take, for example, the third and the second strings from the set, write them down, and get exactly his favorite string.

In the second example Stepan can take, for example, the second, the third and again the second strings from the set and write them down. Then he will get a string "aabaaaab", in which his favorite string "baaab" is a subsequence.

In the third test Stepan can not get his favorite string, because it contains the letter "c", which is not presented in any of the strings in the set.

H. Repairing Of String

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Stepan had a favorite string *s* which consisted of the lowercase letters of the Latin alphabet.

After graduation, he decided to remember it, but it was a long time ago, so he can't now remember it. But Stepan remembers some information about the string, namely the sequence of integers *c*1, *c*2, ..., *cn*, where *n* equals the length of the string *s*, and *ci* equals the number of substrings in the string *s* with the length *i*, consisting of the **same** letters. The substring is a sequence of consecutive characters in the string *s*.

For example, if the Stepan's favorite string is equal to "tttesst", the sequence *c* looks like: *c* = [7, 3, 1, 0, 0, 0, 0].

Stepan asks you to help to repair his favorite string *s* according to the given sequence *c*1, *c*2, ..., *cn*.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 2000) — the length of the Stepan's favorite string.

The second line contains the sequence of integers *c*1, *c*2, ..., *cn* (0 ≤ *ci* ≤ 2000), where *ci* equals the number of substrings of the string *s* with the length *i*, consisting of the same letters.

It is guaranteed that the input data is such that the answer always exists.

**Output**

Print the repaired Stepan's favorite string. If there are several answers, it is allowed to print any of them. The string should contain only lowercase letters of the English alphabet.

**Examples**

**input**

6  
6 3 1 0 0 0

**output**

kkrrrq

**input**

4  
4 0 0 0

**output**

abcd

**Note**

In the first test Stepan's favorite string, for example, can be the string "kkrrrq", because it contains 6 substrings with the length 1, consisting of identical letters (they begin in positions 1, 2, 3, 4, 5 and 6), 3 substrings with the length 2, consisting of identical letters (they begin in positions 1, 3 and 4), and 1 substring with the length 3, consisting of identical letters (it begins in the position 3).

G. Perfectionist Arkadiy

time limit per test

1 second

memory limit per test

256 megabytes

input

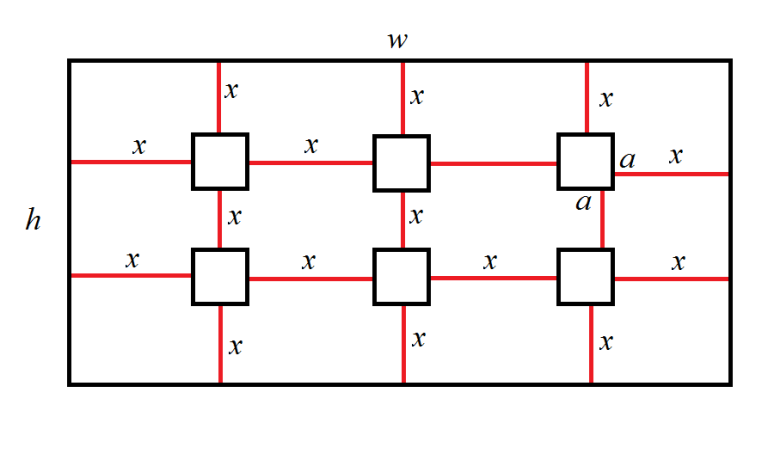
standard input

output

standard output

Arkadiy has lots square photos with size *a* × *a*. He wants to put some of them on a rectangular wall with size *h* × *w*.

The photos which Arkadiy will put on the wall must form a rectangular grid and the distances between neighboring vertically and horizontally photos and also the distances between outside rows and columns of photos to the nearest bound of the wall must be equal to *x*, where *x* is some non-negative **real** number. Look on the picture below for better understanding of the statement.



Arkadiy haven't chosen yet how many photos he would put on the wall, however, he want to put at least one photo. Your task is to determine the **minimum** value of *x* which can be obtained after putting photos, or report that there is no way to put positive number of photos and satisfy all the constraints. Suppose that Arkadiy has enough photos to make any valid arrangement according to the constraints.

Note that Arkadiy wants to put at least one photo on the wall. The photos should not overlap, should completely lie inside the wall bounds and should have sides parallel to the wall sides.

**Input**

The first line contains three integers *a*, *h* and *w* (1 ≤ *a*, *h*, *w* ≤ 109) — the size of photos and the height and the width of the wall.

**Output**

Print one non-negative real number — the minimum value of *x* which can be obtained after putting the photos on the wall. The absolute or the relative error of the answer must not exceed 10- 6.

Print -1 if there is no way to put positive number of photos and satisfy the constraints.

**Examples**

**input**

2 18 13

**output**

0.5

**input**

4 4 4

**output**

0

**input**

3 4 3

**output**

-1

**Note**

In the first example Arkadiy can put 7 rows of photos with 5 photos in each row, so the minimum value of *x* equals to 0.5.

In the second example Arkadiy can put only 1 photo which will take the whole wall, so the minimum value of *x* equals to 0.

In the third example there is no way to put positive number of photos and satisfy the constraints described in the statement, so the answer is -1.

F. Pens And Days Of Week

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Stepan has *n* pens. Every day he uses them, and on the *i*-th day he uses the pen number *i*. On the (*n* + 1)-th day again he uses the pen number 1, on the (*n* + 2)-th — he uses the pen number 2 and so on.

On every working day (from Monday to Saturday, inclusive) Stepan spends exactly 1 milliliter of ink of the pen he uses that day. On Sunday Stepan has a day of rest, he does not stend the ink of the pen he uses that day.

Stepan knows the current volume of ink in each of his pens. Now it's the **Monday morning** and Stepan is going to use the pen **number 1** today. Your task is to determine which pen will run out of ink before all the rest (that is, there will be no ink left in it), if Stepan will use the pens according to the conditions described above.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 50 000) — the number of pens Stepan has.

The second line contains the sequence of integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109), where *ai* is equal to the number of milliliters of ink which the pen number *i* currently has.

**Output**

Print the index of the pen which will run out of ink before all (it means that there will be no ink left in it), if Stepan will use pens according to the conditions described above.

Pens are numbered in the order they are given in input data. The numeration begins from one.

Note that the answer is always unambiguous, since several pens can not end at the same time.

**Examples**

**input**

3  
3 3 3

**output**

2

**input**

5  
5 4 5 4 4

**output**

5

**Note**

In the first test Stepan uses ink of pens as follows:

1. on the day number 1 (Monday) Stepan will use the pen number 1, after that there will be 2 milliliters of ink in it;
2. on the day number 2 (Tuesday) Stepan will use the pen number 2, after that there will be 2 milliliters of ink in it;
3. on the day number 3 (Wednesday) Stepan will use the pen number 3, after that there will be 2 milliliters of ink in it;
4. on the day number 4 (Thursday) Stepan will use the pen number 1, after that there will be 1 milliliters of ink in it;
5. on the day number 5 (Friday) Stepan will use the pen number 2, after that there will be 1 milliliters of ink in it;
6. on the day number 6 (Saturday) Stepan will use the pen number 3, after that there will be 1 milliliters of ink in it;
7. on the day number 7 (Sunday) Stepan will use the pen number 1, but it is a day of rest so he will not waste ink of this pen in it;
8. on the day number 8 (Monday) Stepan will use the pen number 2, after that this pen will run out of ink.

So, the first pen which will not have ink is the pen number 2.

E. Big Number and Remainder

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Stepan has a very big positive integer.

Let's consider all cyclic shifts of Stepan's integer (if we look at his integer like at a string) which are also integers (i.e. they **do not have** leading zeros). Let's call such shifts as *good shifts*. For example, for the integer 10203 the good shifts are the integer itself 10203 and integers 20310 and 31020.

Stepan wants to know the minimum remainder of the division by the given number *m* among all good shifts. Your task is to determine the minimum remainder of the division by *m*.

**Input**

The first line contains the integer which Stepan has. The length of Stepan's integer is between 2 and 200 000 digits, inclusive. It is guaranteed that Stepan's integer does not contain leading zeros.

The second line contains the integer *m* (2 ≤ *m* ≤ 108) — the number by which Stepan divides good shifts of his integer.

**Output**

Print the minimum remainder which Stepan can get if he divides all good shifts of his integer by the given number *m*.

**Examples**

**input**

521  
3

**output**

2

**input**

1001  
5

**output**

0

**input**

5678901234567890123456789  
10000

**output**

123

**Note**

In the first example all good shifts of the integer 521 (good shifts are equal to 521, 215 and 152) has same remainder 2 when dividing by 3.

In the second example there are only two good shifts: the Stepan's integer itself and the shift by one position to the right. The integer itself is 1001 and the remainder after dividing it by 5 equals 1. The shift by one position to the right equals to 1100 and the remainder after dividing it by 5 equals 0, which is the minimum possible remainder.

D. Lie or Truth

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasya has a sequence of cubes and exactly one integer is written on each cube. Vasya exhibited all his cubes in a row. So the sequence of numbers written on the cubes in the order from the left to the right equals to *a*1, *a*2, ..., *an*.

While Vasya was walking, his little brother Stepan played with Vasya's cubes and changed their order, so now the sequence of numbers written on the cubes became equal to *b*1, *b*2, ..., *bn*.

Stepan said that he swapped only cubes which where on the positions between *l* and *r*, inclusive, and did not remove or add any other cubes (i. e. he said that he reordered cubes between positions *l* and *r*, inclusive, in some way).

Your task is to determine if it is possible that Stepan said the truth, or it is guaranteed that Stepan deceived his brother.

**Input**

The first line contains three integers *n*, *l*, *r* (1 ≤ *n* ≤ 105, 1 ≤ *l* ≤ *r* ≤ *n*) — the number of Vasya's cubes and the positions told by Stepan.

The second line contains the sequence *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ *n*) — the sequence of integers written on cubes in the Vasya's order.

The third line contains the sequence *b*1, *b*2, ..., *bn* (1 ≤ *bi* ≤ *n*) — the sequence of integers written on cubes after Stepan rearranged their order.

It is guaranteed that Stepan did not remove or add other cubes, he only rearranged Vasya's cubes.

**Output**

Print "LIE" (without quotes) if it is guaranteed that Stepan deceived his brother. In the other case, print "TRUTH" (without quotes).

**Examples**

**input**

5 2 4  
3 4 2 3 1  
3 2 3 4 1

**output**

TRUTH

**input**

3 1 2  
1 2 3  
3 1 2

**output**

LIE

**input**

4 2 4  
1 1 1 1  
1 1 1 1

**output**

TRUTH

**Note**

In the first example there is a situation when Stepan said the truth. Initially the sequence of integers on the cubes was equal to [3, 4, 2, 3, 1]. Stepan could at first swap cubes on positions 2 and 3 (after that the sequence of integers on cubes became equal to [3, 2, 4, 3, 1]), and then swap cubes in positions 3 and 4 (after that the sequence of integers on cubes became equal to [3, 2, 3, 4, 1]).

In the second example it is not possible that Stepan said truth because he said that he swapped cubes only between positions 1 and 2, but we can see that it is guaranteed that he changed the position of the cube which was on the position 3 at first. So it is guaranteed that Stepan deceived his brother.

In the third example for any values *l* and *r* there is a situation when Stepan said the truth.

C. Maximum Number

time limit per test

1 second

memory limit per test

256 megabytes

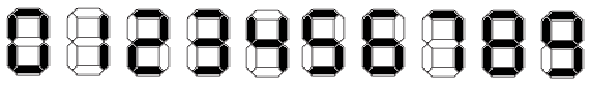
input

standard input

output

standard output

Stepan has the newest electronic device with a display. Different digits can be shown on it. Each digit is shown on a seven-section indicator like it is shown on the picture below.



So, for example, to show the digit 3 on the display, 5 sections must be highlighted; and for the digit 6, 6 sections must be highlighted.

The battery of the newest device allows to highlight at most *n* sections on the display.

Stepan wants to know the maximum possible integer number which can be shown on the display of his newest device. Your task is to determine this number. Note that this number must not contain leading zeros. Assume that the size of the display is enough to show any integer.

**Input**

The first line contains the integer *n* (2 ≤ *n* ≤ 100 000) — the maximum number of sections which can be highlighted on the display.

**Output**

Print the maximum integer which can be shown on the display of Stepan's newest device.

**Examples**

**input**

2

**output**

1

**input**

3

**output**

7

B. Significant Cups

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Stepan is a very experienced olympiad participant. He has *n* cups for Physics olympiads and *m* cups for Informatics olympiads. Each cup is characterized by two parameters — its significance *ci* and width *wi*.

Stepan decided to expose some of his cups on a shelf with width *d* in such a way, that:

* there is at least one Physics cup and at least one Informatics cup on the shelf,
* the total width of the exposed cups does not exceed *d*,
* from each subjects (Physics and Informatics) some of the most significant cups are exposed (i. e. if a cup for some subject with significance *x* is exposed, then all the cups for this subject with significance greater than *x* must be exposed too).

Your task is to determine the maximum possible total significance, which Stepan can get when he exposes cups on the shelf with width *d*, considering all the rules described above. The total significance is the sum of significances of all the exposed cups.

**Input**

The first line contains three integers *n*, *m* and *d* (1 ≤ *n*, *m* ≤ 100 000, 1 ≤ *d* ≤ 109) — the number of cups for Physics olympiads, the number of cups for Informatics olympiads and the width of the shelf.

Each of the following *n* lines contains two integers *ci* and *wi* (1 ≤ *ci*, *wi* ≤ 109) — significance and width of the *i*-th cup for Physics olympiads.

Each of the following *m* lines contains two integers *cj* and *wj* (1 ≤ *cj*, *wj* ≤ 109) — significance and width of the *j*-th cup for Informatics olympiads.

**Output**

Print the maximum possible total significance, which Stepan can get exposing cups on the shelf with width *d*, considering all the rules described in the statement.

If there is no way to expose cups on the shelf, then print 0.

**Examples**

**input**

3 1 8  
4 2  
5 5  
4 2  
3 2

**output**

8

**input**

4 3 12  
3 4  
2 4  
3 5  
3 4  
3 5  
5 2  
3 4

**output**

11

**input**

2 2 2  
5 3  
6 3  
4 2  
8 1

**output**

0

**Note**

In the first example Stepan has only one Informatics cup which must be exposed on the shelf. Its significance equals 3 and width equals 2, so after Stepan exposes it, the width of free space on the shelf becomes equal to 6. Also, Stepan must expose the second Physics cup (which has width 5), because it is the most significant cup for Physics (its significance equals 5). After that Stepan can not expose more cups on the shelf, because there is no enough free space. Thus, the maximum total significance of exposed cups equals to 8.

A. Amusement Park

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Pupils decided to go to amusement park. Some of them were with parents. In total, *n* people came to the park and they all want to get to the most extreme attraction and roll on it exactly **once**.

Tickets for group of *x* people are sold on the attraction, there should be at least one adult in each group (it is possible that the group consists of one adult). The ticket price for such group is *c*1 + *c*2·(*x* - 1)2 (in particular, if the group consists of one person, then the price is *c*1).

All pupils who came to the park and their parents decided to split into groups in such a way that each visitor join exactly one group, and the total price of visiting the most extreme attraction is as low as possible. You are to determine this minimum possible total price. There should be at least one adult in each group.

**Input**

The first line contains three integers *n*, *c*1 and *c*2 (1 ≤ *n* ≤ 200 000, 1 ≤ *c*1, *c*2 ≤ 107) — the number of visitors and parameters for determining the ticket prices for a group.

The second line contains the string of length *n*, which consists of zeros and ones. If the *i*-th symbol of the string is zero, then the *i*-th visitor is a pupil, otherwise the *i*-th person is an adult. It is guaranteed that there is at least one adult. It is possible that there are no pupils.

**Output**

Print the minimum price of visiting the most extreme attraction for all pupils and their parents. Each of them should roll on the attraction exactly once.

**Examples**

**input**

3 4 1  
011

**output**

8

**input**

4 7 2  
1101

**output**

18

**Note**

In the first test one group of three people should go to the attraction. Then they have to pay 4 + 1 \* (3 - 1)2 = 8.

In the second test it is better to go to the attraction in two groups. The first group should consist of two adults (for example, the first and the second person), the second should consist of one pupil and one adult (the third and the fourth person). Then each group will have a size of two and for each the price of ticket is 7 + 2 \* (2 - 1)2 = 9. Thus, the total price for two groups is 18.

F. Test Data Generation

time limit per test

5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Test data generation is not an easy task! Often, generating big random test cases is not enough to ensure thorough testing of solutions for correctness.

For example, consider a problem from an old Codeforces round. Its input format looks roughly as follows:

*The first line contains a single integer n (1 ≤ n ≤ maxn) — the size of the set. The second line contains n distinct integers a1, a2, ..., an(1 ≤ ai ≤ maxa) — the elements of the set****in increasing order****.*

If you don't pay attention to the problem solution, it looks fairly easy to generate a good test case for this problem. Let *n* = *maxn*, take random distinct *ai* from 1 to *maxa*, sort them... Soon you understand that it's not that easy.

Here is the actual problem solution. Let *g* be the greatest common divisor of *a*1, *a*2, ..., *an*. Let *x* = *an* / *g* - *n*. Then the correct solution outputs "Alice" if *x* is odd, and "Bob" if *x* is even.

Consider two wrong solutions to this problem which differ from the correct one only in the formula for calculating *x*.

The first wrong solution calculates *x* as *x* = *an* / *g* (without subtracting *n*).

The second wrong solution calculates *x* as *x* = *an* - *n* (without dividing by *g*).

A test case is interesting if it makes **both** wrong solutions output an incorrect answer.

Given *maxn*, *maxa* and *q*, find the number of interesting test cases satisfying the constraints, and output it modulo *q*.

**Input**

The only line contains three integers *maxn*, *maxa* and *q* (1 ≤ *maxn* ≤ 30 000; *maxn* ≤ *maxa* ≤ 109; 104 ≤ *q* ≤ 105 + 129).

**Output**

Output a single integer — the number of test cases which satisfy the constraints and make both wrong solutions output an incorrect answer, modulo *q*.

**Examples**

**input**

3 6 100000

**output**

4

**input**

6 21 100129

**output**

154

**input**

58 787788 50216

**output**

46009

**Note**

In the first example, interesting test cases look as follows:

1 1 1 3  
2 4 6 2 4 6

E. Blog Post Rating

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

It's well-known that blog posts are an important part of Codeforces platform. Every blog post has a global characteristic changing over time — its *community rating*. A newly created blog post's community rating is 0. Codeforces users may visit the blog post page and rate it, changing its community rating by +1 or -1.

Consider the following model of Codeforces users' behavior. The *i*-th user has his own *estimated blog post rating* denoted by an integer *ai*. When a user visits a blog post page, he compares his estimated blog post rating to its community rating. If his estimated rating is higher, he rates the blog post with +1 (thus, the blog post's community rating increases by 1). If his estimated rating is lower, he rates the blog post with -1 (decreasing its community rating by 1). If the estimated rating and the community rating are equal, user doesn't rate the blog post at all (in this case we'll say that user rates the blog post for 0). In any case, after this procedure user closes the blog post page and never opens it again.

Consider a newly created blog post with the initial community rating of 0. For each of *n* Codeforces users, numbered from 1 to *n*, his estimated blog post rating *ai* is known.

For each *k* from 1 to *n*, inclusive, the following question is asked. Let users with indices from 1 to *k*, **in some order**, visit the blog post page, rate the blog post and close the page. Each user opens the blog post only after the previous user closes it. What could be the maximum possible community rating of the blog post after these *k* visits?

**Input**

The first line contains a single integer *n* (1 ≤ *n* ≤ 5·105) — the number of Codeforces users.

The second line contains *n* integers *a*1, *a*2, ..., *an* ( - 5·105 ≤ *ai* ≤ 5·105) — estimated blog post ratings for users in order from 1 to *n*.

**Output**

For each *k* from 1 to *n*, output a single integer equal to the maximum possible community rating of the blog post after users with indices from 1 to *k*, in some order, visit the blog post page, rate the blog post, and close the page.

**Examples**

**input**

4  
2 0 2 2

**output**

1  
1  
2  
2

**input**

7  
2 -3 -2 5 0 -3 1

**output**

1  
0  
-1  
0  
1  
1  
2

D. Perishable Roads

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

In the country of Never, there are *n* cities and a well-developed road system. There is exactly one bidirectional road between every pair of cities, thus, there are as many as http://codeforces.com/predownloaded/31/80/3180a33fa5d6fc9ced6f37180d95ed4545de8035.png roads! No two roads intersect, and no road passes through intermediate cities. The art of building tunnels and bridges has been mastered by Neverians.

An independent committee has evaluated each road of Never with a positive integer called the *perishability* of the road. The lower the road's perishability is, the more pleasant it is to drive through this road.

It's the year of transport in Never. It has been decided to build a museum of transport in one of the cities, and to set a single signpost directing to some city (not necessarily the one with the museum) in each of the other cities. The signposts must satisfy the following important condition: if any Neverian living in a city without the museum starts travelling from that city following the directions of the signposts, then this person will eventually arrive in the city with the museum.

Neverians are incredibly positive-minded. If a Neverian travels by a route consisting of several roads, he considers the *perishability of the route* to be equal to the smallest perishability of all the roads in this route.

The government of Never has not yet decided where to build the museum, so they consider all *n* possible options. The most important is the sum of perishabilities of the routes to the museum city from all the other cities of Never, if the travelers strictly follow the directions of the signposts. The government of Never cares about their citizens, so they want to set the signposts in a way which minimizes this sum. Help them determine the minimum possible sum for all *n* possible options of the city where the museum can be built.

**Input**

The first line contains a single integer *n* (2 ≤ *n* ≤ 2000) — the number of cities in Never.

The following *n* - 1 lines contain the description of the road network. The *i*-th of these lines contains *n* - *i* integers. The *j*-th integer in the *i*-th line denotes the perishability of the road between cities *i* and *i* + *j*.

All road perishabilities are between 1 and 109, inclusive.

**Output**

For each city in order from 1 to *n*, output the minimum possible sum of perishabilities of the routes to this city from all the other cities of Never if the signposts are set in a way which minimizes this sum.

**Examples**

**input**

3  
1 2  
3

**output**

2  
2  
3

**input**

6  
2 9 9 6 6  
7 1 9 10  
9 2 5  
4 10  
8

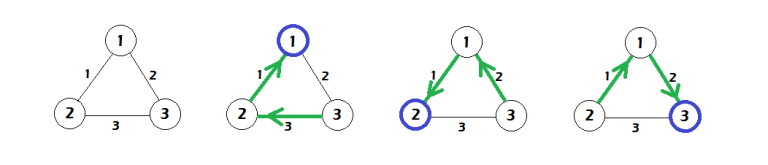
**output**

6  
5  
7  
5  
7  
11

**Note**

The first example is explained by the picture below. From left to right, there is the initial road network and the optimal directions of the signposts in case the museum is built in city 1, 2 and 3, respectively. The museum city is represented by a blue circle, the directions of the signposts are represented by green arrows.

For instance, if the museum is built in city 3, then the signpost in city 1 must be directed to city 3, while the signpost in city 2 must be directed to city 1. Then the route from city 1 to city 3 will have perishability 2, while the route from city 2 to city 3 will have perishability 1. The sum of perishabilities of these routes is 3.



C. Prairie Partition

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

It can be shown that any positive integer *x* can be uniquely represented as *x* = 1 + 2 + 4 + ... + 2*k*- 1 + *r*, where *k* and *r* are integers, *k* ≥ 0, 0 < *r* ≤ 2*k*. Let's call that representation *prairie partition* of *x*.

For example, the prairie partitions of 12, 17, 7 and 1 are:

12 = 1 + 2 + 4 + 5,

17 = 1 + 2 + 4 + 8 + 2,

7 = 1 + 2 + 4,

1 = 1.

Alice took a sequence of positive integers (possibly with repeating elements), replaced every element with the sequence of summands in its prairie partition, arranged the resulting numbers in non-decreasing order and gave them to Borys. Now Borys wonders how many elements Alice's original sequence could contain. Find all possible options!

**Input**

The first line contains a single integer *n* (1 ≤ *n* ≤ 105) — the number of numbers given from Alice to Borys.

The second line contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 1012; *a*1 ≤ *a*2 ≤ ... ≤ *an*) — the numbers given from Alice to Borys.

**Output**

Output, **in increasing order**, all possible values of *m* such that there exists a sequence of positive integers of length *m* such that if you replace every element with the summands in its prairie partition and arrange the resulting numbers in non-decreasing order, you will get the sequence given in the input.

If there are no such values of *m*, output a single integer -1.

**Examples**

**input**

8  
1 1 2 2 3 4 5 8

**output**

2

**input**

6  
1 1 1 2 2 2

**output**

2 3

**input**

5  
1 2 4 4 4

**output**

-1

**Note**

In the first example, Alice could get the input sequence from [6, 20] as the original sequence.

In the second example, Alice's original sequence could be either [4, 5] or [3, 3, 3].

B. Dynamic Problem Scoring

time limit per test

2 seconds

memory limit per test

256 megabytes

input

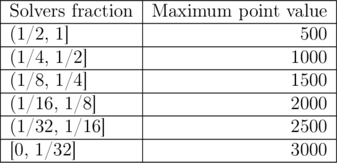
standard input

output

standard output

Vasya and Petya take part in a Codeforces round. The round lasts for two hours and contains five problems.

For this round the dynamic problem scoring is used. If you were lucky not to participate in any Codeforces round with dynamic problem scoring, here is what it means. The maximum point value of the problem depends on the ratio of the number of participants who solved the problem to the total number of round participants. Everyone who made at least one submission is considered to be participating in the round.



Pay attention to the range bounds. For example, if 40 people are taking part in the round, and 10 of them solve a particular problem, then the solvers fraction is equal to 1 / 4, and the problem's maximum point value is equal to 1500.

If the problem's maximum point value is equal to *x*, then for each whole minute passed from the beginning of the contest to the moment of the participant's correct submission, the participant loses *x* / 250 points. For example, if the problem's maximum point value is 2000, and the participant submits a correct solution to it 40 minutes into the round, this participant will be awarded with 2000·(1 - 40 / 250) = 1680 points for this problem.

There are *n* participants in the round, including Vasya and Petya. For each participant and each problem, the number of minutes which passed between the beginning of the contest and the submission of this participant to this problem is known. It's also possible that this participant made no submissions to this problem.

With two seconds until the end of the round, all participants' submissions have passed pretests, and not a single hack attempt has been made. Vasya believes that no more submissions or hack attempts will be made in the remaining two seconds, and every submission will pass the system testing.

Unfortunately, Vasya is a cheater. He has registered 109 + 7 new accounts for the round. Now Vasya can submit any of his solutions from these new accounts in order to change the maximum point values of the problems. Vasya can also submit any wrong solutions to any problems. Note that Vasya can not submit correct solutions to the problems he hasn't solved.

Vasya seeks to score strictly more points than Petya in the current round. Vasya has already prepared the scripts which allow to obfuscate his solutions and submit them into the system from any of the new accounts in just fractions of seconds. However, Vasya doesn't want to make his cheating too obvious, so he wants to achieve his goal while making submissions from the smallest possible number of new accounts.

Find the smallest number of new accounts Vasya needs in order to beat Petya (provided that Vasya's assumptions are correct), or report that Vasya can't achieve his goal.

**Input**

The first line contains a single integer *n* (2 ≤ *n* ≤ 120) — the number of round participants, including Vasya and Petya.

Each of the next *n* lines contains five integers *ai*, 1, *ai*, 2..., *ai*, 5 ( - 1 ≤ *ai*,*j* ≤ 119) — the number of minutes passed between the beginning of the round and the submission of problem *j* by participant *i*, or -1 if participant *i* hasn't solved problem *j*.

It is guaranteed that each participant has made at least one successful submission.

Vasya is listed as participant number 1, Petya is listed as participant number 2, all the other participants are listed in no particular order.

**Output**

Output a single integer — the number of new accounts Vasya needs to beat Petya, or -1 if Vasya can't achieve his goal.

**Examples**

**input**

2  
5 15 40 70 115  
50 45 40 30 15

**output**

2

**input**

3  
55 80 10 -1 -1  
15 -1 79 60 -1  
42 -1 13 -1 -1

**output**

3

**input**

5  
119 119 119 119 119  
0 0 0 0 -1  
20 65 12 73 77  
78 112 22 23 11  
1 78 60 111 62

**output**

27

**input**

4  
-1 20 40 77 119  
30 10 73 50 107  
21 29 -1 64 98  
117 65 -1 -1 -1

**output**

-1

**Note**

In the first example, Vasya's optimal strategy is to submit the solutions to the last three problems from two new accounts. In this case the first two problems will have the maximum point value of 1000, while the last three problems will have the maximum point value of 500. Vasya's score will be equal to 980 + 940 + 420 + 360 + 270 = 2970 points, while Petya will score just 800 + 820 + 420 + 440 + 470 = 2950 points.

In the second example, Vasya has to make a single unsuccessful submission to any problem from two new accounts, and a single successful submission to the first problem from the third new account. In this case, the maximum point values of the problems will be equal to 500, 1500, 1000, 1500, 3000. Vasya will score 2370 points, while Petya will score just 2294 points.

In the third example, Vasya can achieve his goal by submitting the solutions to the first four problems from 27 new accounts. The maximum point values of the problems will be equal to 500, 500, 500, 500, 2000. Thanks to the high cost of the fifth problem, Vasya will manage to beat Petya who solved the first four problems very quickly, but couldn't solve the fifth one.

A. Success Rate

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are an experienced Codeforces user. Today you found out that during your activity on Codeforces you have made *y* submissions, out of which *x* have been successful. Thus, your current success rate on Codeforces is equal to *x* / *y*.

Your favorite rational number in the [0;1] range is *p* / *q*. Now you wonder: what is the smallest number of submissions you have to make if you want your success rate to be *p* / *q*?

**Input**

The first line contains a single integer *t* (1 ≤ *t* ≤ 1000) — the number of test cases.

Each of the next *t* lines contains four integers *x*, *y*, *p* and *q* (0 ≤ *x* ≤ *y* ≤ 109; 0 ≤ *p* ≤ *q* ≤ 109; *y* > 0; *q* > 0).

It is guaranteed that *p* / *q* is an irreducible fraction.

**Hacks.** For hacks, an additional constraint of *t* ≤ 5 must be met.

**Output**

For each test case, output a single integer equal to the smallest number of submissions you have to make if you want your success rate to be equal to your favorite rational number, or -1 if this is impossible to achieve.

**Example**

**input**

4  
3 10 1 2  
7 14 3 8  
20 70 2 7  
5 6 1 1

**output**

4  
10  
0  
-1

**Note**

In the first example, you have to make 4 successful submissions. Your success rate will be equal to 7 / 14, or 1 / 2.

In the second example, you have to make 2 successful and 8 unsuccessful submissions. Your success rate will be equal to 9 / 24, or 3 / 8.

In the third example, there is no need to make any new submissions. Your success rate is already equal to 20 / 70, or 2 / 7.

In the fourth example, the only unsuccessful submission breaks your hopes of having the success rate equal to 1.

E. Verifying Kingdom

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

**This is an interactive problem.**

The judge has a hidden rooted full binary tree with *n* leaves. A full binary tree is one where every node has either 0 or 2 children. The nodes with 0 children are called the leaves of the tree. Since this is a full binary tree, there are exactly 2*n* - 1 nodes in the tree. The leaves of the judge's tree has labels from 1 to *n*. You would like to reconstruct a tree that is isomorphic to the judge's tree. To do this, you can ask some questions.

A question consists of printing the label of three distinct leaves *a*1, *a*2, *a*3. Let the *depth* of a node be the shortest distance from the node to the root of the tree. Let *LCA*(*a*, *b*) denote the node with maximum depth that is a common ancestor of the nodes *a* and *b*.

Consider *X* = *LCA*(*a*1, *a*2), *Y* = *LCA*(*a*2, *a*3), *Z* = *LCA*(*a*3, *a*1). The judge will tell you which one of *X*, *Y*, *Z* has the maximum depth. Note, this pair is uniquely determined since the tree is a binary tree; there can't be any ties.

More specifically, if *X* (or *Y*, *Z* respectively) maximizes the depth, the judge will respond with the string "X" (or "Y", "Z" respectively).

You may only ask at most 10·*n* questions.

**Input**

The first line of input will contain a single integer *n* (3 ≤ *n* ≤ 1 000) — the number of leaves in the tree.

**Output**

To print the final answer, print out the string "-1" on its own line. Then, the next line should contain 2*n* - 1 integers. The *i*-th integer should be the parent of the *i*-th node, or -1, if it is the root.

Your answer will be judged correct if your output is isomorphic to the judge's tree. In particular, the labels of the leaves do not need to be labeled from 1 to *n*. Here, isomorphic means that there exists a permutation π such that node *i* is the parent of node *j* in the judge tree if and only node π(*i*) is the parent of node π(*j*) in your tree.

**Interaction**

To ask a question, print out three distinct integers *a*1, *a*2, *a*3. These integers should be between 1 and *n*, inclusive.

The judge will respond with a single character, either "X", "Y", "Z".

If the string is "X" (or "Y", "Z" respectively), that means the pair (*a*1, *a*2) (or (*a*2, *a*3), (*a*3, *a*1) respectively) has the deepest *LCA* among the three pairs.

You may only ask a question at most 10·*n* times, otherwise, you will get Wrong Answer.

When you are ready to answer, print out a single integer "-1" on its own line. The next line should contain 2*n* - 1 integers. The *i*-th integer should be the parent of the *i*-th node, or -1, if it is the root. Do not forget to flush the final answer as well. Printing the answer does not count as asking a question.

You will get Wrong Answer verdict if

* Your question or answers are not in the format described in this statement.
* You ask strictly more than 10·*n* questions.
* Your question contains duplicate indices.
* Your final answer is not isomorphic to the judge tree.

You will get Idleness Limit Exceeded if you don't print anything or if you forget to flush the output, including for the final answer (more info about flushing output below).

To flush you can use (just after printing an integer and end-of-line):

* fflush(stdout) in C++;
* System.out.flush() in Java;
* stdout.flush() in Python;
* flush(output) in Pascal;
* See the documentation for other languages.

If at any moment your program reads -1 as an answer, it should immediately exit normally (for example, by calling exit(0)). You will get Wrong Answer in this case, it means that you made more queries than allowed, or made an invalid query. If you ignore this, you can get other verdicts since your program will continue to read from a closed stream.

**Hacking** To hack someone, use the following format

n  
p\_1 p\_2 ... p\_{2n-1}

This denotes a tree where the parent of the *i*-th node is *pi* (*pi* =  - 1 or *n* < *pi* ≤ 2*n* - 1). If *pi* is equal to -1, then node *i* is the root. This input must describe a valid full rooted binary tree.

Of course, contestant programs will not be able to see this input.

**Example**

**input**

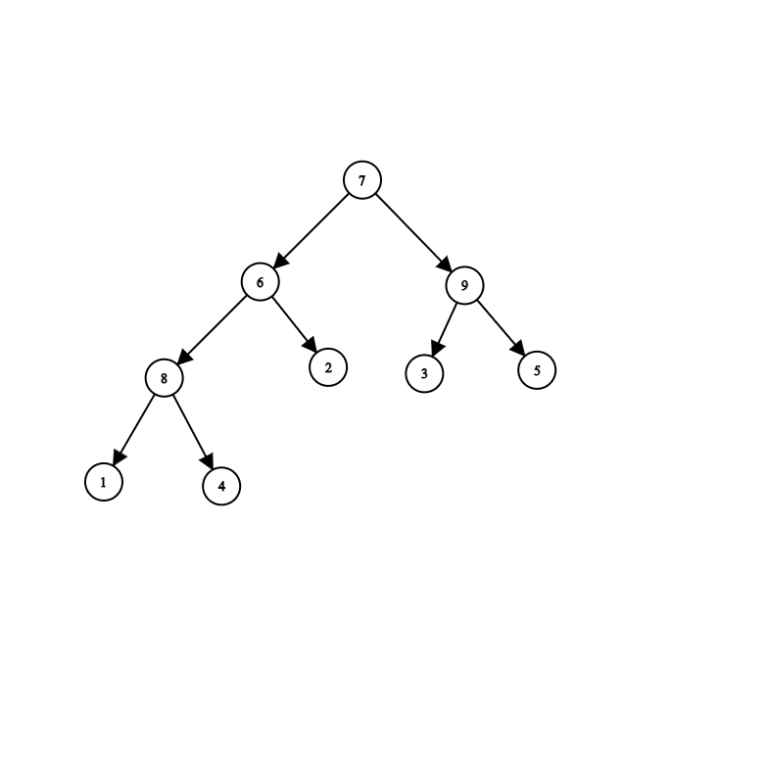
5  
X  
Z  
Y  
Y  
X

**output**

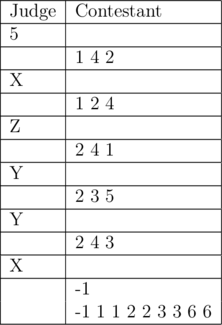
1 4 2  
1 2 4  
2 4 1  
2 3 5  
2 4 3  
-1  
-1 1 1 2 2 3 3 6 6

**Note**

For the first sample, the judge has the hidden tree:



Here is a more readable format of the interaction:



The last line can also be 8 6 9 8 9 7 -1 6 7.

D. Varying Kibibits

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given *n* integers *a*1, *a*2, ..., *an*. Denote this list of integers as *T*.

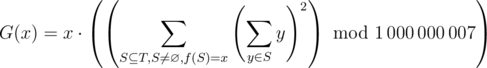
Let *f*(*L*) be a function that takes in a non-empty list of integers *L*.

The function will output another integer as follows:

* First, all integers in *L* are padded with leading zeros so they are all the same length as the maximum length number in *L*.
* We will construct a string where the *i*-th character is the minimum of the *i*-th character in padded input numbers.
* The output is the number representing the string interpreted in base 10.

For example *f*(10, 9) = 0, *f*(123, 321) = 121, *f*(530, 932, 81) = 30.

Define the function



Here, http://codeforces.com/predownloaded/ca/53/ca5303a37ff006d269018f2a89879903614aea29.png denotes a subsequence.

In other words, *G*(*x*) is the sum of squares of sum of elements of nonempty subsequences of *T* that evaluate to *x* when plugged into *f*modulo 1 000 000 007, then multiplied by *x*. The last multiplication is not modded.

You would like to compute *G*(0), *G*(1), ..., *G*(999 999). To reduce the output size, print the value http://codeforces.com/predownloaded/13/0b/130b9a6add9741acbc5e33e9998a7ccd0c6d466d.png, where http://codeforces.com/predownloaded/7b/ea/7beade55e90846d70020a3d03521d3458b66751b.png denotes the bitwise XOR operator.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 1 000 000) — the size of list *T*.

The next line contains *n* space-separated integers, *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 999 999) — the elements of the list.

**Output**

Output a single integer, the answer to the problem.

**Examples**

**input**

3  
123 321 555

**output**

292711924

**input**

1  
999999

**output**

997992010006992

**input**

10  
1 1 1 1 1 1 1 1 1 1

**output**

28160

**Note**

For the first sample, the nonzero values of *G* are *G*(121) = 144 611 577, *G*(123) = 58 401 999, *G*(321) = 279 403 857, *G*(555) = 170 953 875. The bitwise XOR of these numbers is equal to 292 711 924.

For example, http://codeforces.com/predownloaded/82/a3/82a3d4f5778262f1fd83034e369c61faccfb732d.png, since the subsequences [123] and [123, 555] evaluate to 123 when plugged into *f*.

For the second sample, we have http://codeforces.com/predownloaded/24/42/24428649fedd810f84fa5a1ece16b203c5e509d0.png

For the last sample, we have http://codeforces.com/predownloaded/e5/e1/e5e14ecb1d7f48680c2fcab9f9b5dae74cd479af.png, where http://codeforces.com/predownloaded/7d/77/7d77fbe8561748fcb242b79fe32124375da40693.png is the binomial coefficient.

C. Vulnerable Kerbals

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given an integer *m*, and a list of *n* distinct integers between 0 and *m* - 1.

You would like to construct a sequence satisfying the properties:

* Each element is an integer between 0 and *m* - 1, inclusive.
* All prefix products of the sequence modulo *m* are distinct.
* No prefix product modulo *m* appears as an element of the input list.
* The length of the sequence is maximized.

Construct any sequence satisfying the properties above.

**Input**

The first line of input contains two integers *n* and *m* (0 ≤ *n* < *m* ≤ 200 000) — the number of forbidden prefix products and the modulus.

If *n* is non-zero, the next line of input contains *n* distinct integers between 0 and *m* - 1, the forbidden prefix products. If *n* is zero, this line doesn't exist.

**Output**

On the first line, print the number *k*, denoting the length of your sequence.

On the second line, print *k* space separated integers, denoting your sequence.

**Examples**

**input**

0 5

**output**

5  
1 2 4 3 0

**input**

3 10  
2 9 1

**output**

6  
3 9 2 9 8 0

**Note**

For the first case, the prefix products of this sequence modulo *m* are [1, 2, 3, 4, 0].

For the second case, the prefix products of this sequence modulo *m* are [3, 7, 4, 6, 8, 0].

B. Volatile Kite

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a convex polygon *P* with *n* distinct vertices *p*1, *p*2, ..., *pn*. Vertex *pi* has coordinates (*xi*, *yi*) in the 2D plane. These vertices are listed in clockwise order.

You can choose a real number *D* and move each vertex of the polygon a distance of at most *D* from their original positions.

Find the maximum value of *D* such that no matter how you move the vertices, the polygon does not intersect itself and stays convex.

**Input**

The first line has one integer *n* (4 ≤ *n* ≤ 1 000) — the number of vertices.

The next *n* lines contain the coordinates of the vertices. Line *i* contains two integers *xi* and *yi* ( - 109 ≤ *xi*, *yi* ≤ 109) — the coordinates of the *i*-th vertex. These points are guaranteed to be given in clockwise order, and will form a strictly convex polygon (in particular, no three consecutive points lie on the same straight line).

**Output**

Print one real number *D*, which is the maximum real number such that no matter how you move the vertices, the polygon stays convex.

Your answer will be considered correct if its absolute or relative error does not exceed 10- 6.

Namely, let's assume that your answer is *a* and the answer of the jury is *b*. The checker program will consider your answer correct if http://codeforces.com/predownloaded/c6/2e/c62ea64d4651240724c5ac4779b671c741edec24.png.

**Examples**

**input**

4  
0 0  
0 1  
1 1  
1 0

**output**

0.3535533906

**input**

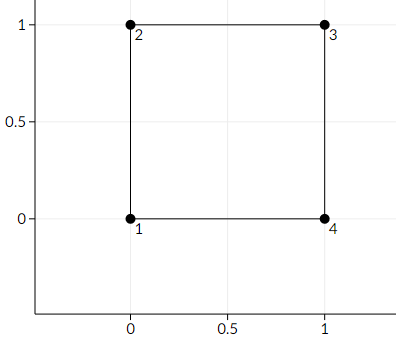
6  
5 0  
10 0  
12 -4  
10 -8  
5 -8  
3 -4

**output**

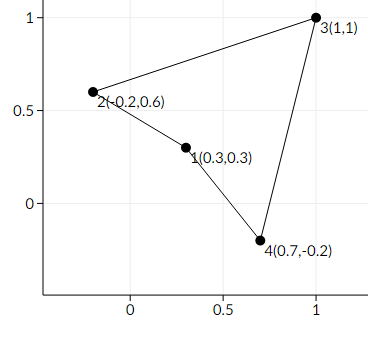
1.0000000000

**Note**

Here is a picture of the first sample



Here is an example of making the polygon non-convex.



This is not an optimal solution, since the maximum distance we moved one point is  ≈ 0.4242640687, whereas we can make it non-convex by only moving each point a distance of at most  ≈ 0.3535533906.

A. Voltage Keepsake

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You have *n* devices that you want to use simultaneously.

The *i*-th device uses *ai* units of power per second. This usage is continuous. That is, in λ seconds, the device will use λ·*ai* units of power. The *i*-th device currently has *bi* units of power stored. All devices can store an arbitrary amount of power.

You have a single charger that can plug to any single device. The charger will add *p* units of power per second to a device. This charging is continuous. That is, if you plug in a device for λ seconds, it will gain λ·*p* units of power. You can switch which device is charging at any arbitrary unit of time (including real numbers), and the time it takes to switch is negligible.

You are wondering, what is the maximum amount of time you can use the devices until one of them hits 0 units of power.

If you can use the devices indefinitely, print -1. Otherwise, print the maximum amount of time before any one device hits 0 power.

**Input**

The first line contains two integers, *n* and *p* (1 ≤ *n* ≤ 100 000, 1 ≤ *p* ≤ 109) — the number of devices and the power of the charger.

This is followed by *n* lines which contain two integers each. Line *i* contains the integers *ai* and *bi* (1 ≤ *ai*, *bi* ≤ 100 000) — the power of the device and the amount of power stored in the device in the beginning.

**Output**

If you can use the devices indefinitely, print -1. Otherwise, print the maximum amount of time before any one device hits 0 power.

Your answer will be considered correct if its absolute or relative error does not exceed 10- 4.

Namely, let's assume that your answer is *a* and the answer of the jury is *b*. The checker program will consider your answer correct if http://codeforces.com/predownloaded/c8/be/c8be6ce47c501ca6c4f051913a05ec6e56c86b13.png.

**Examples**

**input**

2 1  
2 2  
2 1000

**output**

2.0000000000

**input**

1 100  
1 1

**output**

-1

**input**

3 5  
4 3  
5 2  
6 1

**output**

0.5000000000

**Note**

In sample test 1, you can charge the first device for the entire time until it hits zero power. The second device has enough power to last this time without being charged.

In sample test 2, you can use the device indefinitely.

In sample test 3, we can charge the third device for 2 / 5 of a second, then switch to charge the second device for a 1 / 10 of a second.

F. Bear and Isomorphic Points

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Bearland is a big square on the plane. It contains all points with coordinates not exceeding 106 by the absolute value.

There are *n* houses in Bearland. The *i*-th of them is located at the point (*xi*, *yi*). The *n* points are distinct, but some subsets of them may be collinear.

Bear Limak lives in the first house. He wants to destroy his house and build a new one somewhere in Bearland.

Bears don't like big changes. For every three points (houses) *pi*, *pj* and *pk*, the sign of their cross product (*pj* - *pi*) × (*pk* - *pi*) should be the same before and after the relocation. If it was negative/positive/zero, it should still be negative/positive/zero respectively. This condition should be satisfied for all triples of indices (*i*, *j*, *k*), possibly equal to each other or different than 1. Additionally, Limak isn't allowed to build the house at the point where some other house already exists (but it can be the point where his old house was).

In the formula above, we define the difference and the cross product of points (*ax*, *ay*) and (*bx*, *by*) as:

(*ax*, *ay*) - (*bx*, *by*) = (*ax* - *bx*, *ay* - *by*),

(*ax*, *ay*) × (*bx*, *by*) = *ax*·*by* - *ay*·*bx*.

Consider a set of possible new placements of Limak's house. Your task is to find the area of that set of points.

Formally, let's say that Limak chooses the new placement randomly (each coordinate is chosen independently uniformly at random from the interval [ - 106, 106]). Let *p* denote the probability of getting the allowed placement of new house. Let *S* denote the area of Bearland (*S* = 4·1012). Your task is to find *p*·*S*.

**Input**

The first line of the input contains an integer *T* (1 ≤ *T* ≤ 500) — the number of test cases. The description of the test cases follows.

The first line of the description of a test case contains an integer *n* (3 ≤ *n* ≤ 200 000) — the number of houses.

The *i*-th of the next *n* lines contains two integers *xi* and *yi* ( - 106 ≤ *xi*, *yi* ≤ 106) — coordinates of the *i*-th house. No two houses are located at the same point in the same test case. Limak lives in the first house.

The sum of *n* won't exceed 200 000.

**Output**

Print one real value, denoting the area of the set of points that are possible new placements of Limak's house.

Your answer will be considered correct if its absolute or relative error doesn't exceed 10- 6. More precisely, let the jury's answer be *b*, and your answer be *a*. Then your answer will be accepted if and only if http://codeforces.com/predownloaded/68/88/6888d376f2feb6fac73661dd624e9baafd7cd851.png.

**Example**

**input**

4  
4  
5 3  
0 1  
10 1  
3 51  
3  
-999123 700000  
-950000 123456  
-950000 987654  
3  
2 3  
10 -1  
-4 6  
5  
1 3  
5 2  
6 1  
4 4  
-3 3

**output**

250.000000000000  
100000000000.000000000000  
0.000000000000  
6.562500000000

**Note**

In the sample test, there are 4 test cases.

In the first test case, there are four houses and Limak's one is in (5, 3). The set of valid new placements form a triangle with vertices in points (0, 1), (10, 1) and (3, 51), without its sides. The area of such a triangle is 250.

In the second test case, the set of valid new placements form a rectangle of width 50 000 and height 2 000 000. Don't forget that the new placement must be inside the big square that represents Bearland.

In the third test case, the three given points are collinear. Each cross product is equal to 0 and it should be 0 after the relocation as well. Hence, Limak's new house must lie on the line that goes through the given points. Since it must also be inside the big square, new possible placements are limited to some segment (excluding the two points where the other houses are). The area of any segment is 0.

E. Bear and Rectangle Strips

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Limak has a grid that consists of 2 rows and *n* columns. The *j*-th cell in the *i*-th row contains an integer *ti*,*j* which can be positive, negative or zero.

A non-empty rectangle of cells is called *nice* if and only if the sum of numbers in its cells is equal to 0.

Limak wants to choose some nice rectangles and give them to his friends, as gifts. No two chosen rectangles should share a cell. What is the maximum possible number of nice rectangles Limak can choose?

**Input**

The first line of the input contains an integer *n* (1 ≤ *n* ≤ 300 000) — the number of columns in the grid.

The next two lines contain numbers in the grid. The *i*-th of those two lines contains *n* integers *ti*, 1, *ti*, 2, ..., *ti*,*n* ( - 109 ≤ *ti*,*j* ≤ 109).

**Output**

Print one integer, denoting the maximum possible number of cell-disjoint nice rectangles.

**Examples**

**input**

6  
70 70 70 70 70 -15  
90 -60 -30 30 -30 15

**output**

3

**input**

4  
0 -1 0 0  
0 0 1 0

**output**

6

**input**

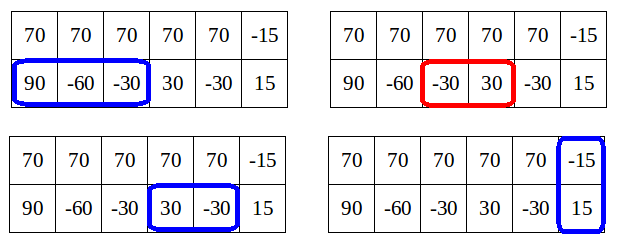
3  
1000000000 999999999 -1000000000  
999999999 -1000000000 -999999998

**output**

1

**Note**

In the first sample, there are four nice rectangles:



Limak can't choose all of them because they are not disjoint. He should take three nice rectangles: those denoted as blue frames on the drawings.

In the second sample, it's optimal to choose six nice rectangles, each consisting of one cell with a number 0.

In the third sample, the only nice rectangle is the whole grid — the sum of all numbers is 0. Clearly, Limak can choose at most one nice rectangle, so the answer is 1.

D. Bear and Company

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Bear Limak prepares problems for a programming competition. Of course, it would be unprofessional to mention the sponsor name in the statement. Limak takes it seriously and he is going to change some words. To make it still possible to read, he will try to modify each word as little as possible.

Limak has a string *s* that consists of uppercase English letters. In one move he can swap two **adjacent** letters of the string. For example, he can transform a string "ABBC" into "BABC" or "ABCB" in one move.

Limak wants to obtain a string without a substring "VK" (i.e. there should be no letter 'V' immediately followed by letter 'K'). It can be easily proved that it's possible for any initial string *s*.

What is the minimum possible number of moves Limak can do?

**Input**

The first line of the input contains an integer *n* (1 ≤ *n* ≤ 75) — the length of the string.

The second line contains a string *s*, consisting of uppercase English letters. The length of the string is equal to *n*.

**Output**

Print one integer, denoting the minimum possible number of moves Limak can do, in order to obtain a string without a substring "VK".

**Examples**

**input**

4  
VKVK

**output**

3

**input**

5  
BVVKV

**output**

2

**input**

7  
VVKEVKK

**output**

3

**input**

20  
VKVKVVVKVOVKVQKKKVVK

**output**

8

**input**

5  
LIMAK

**output**

0

**Note**

In the first sample, the initial string is "VKVK". The minimum possible number of moves is 3. One optimal sequence of moves is:

1. Swap two last letters. The string becomes "VKKV".
2. Swap first two letters. The string becomes "KVKV".
3. Swap the second and the third letter. The string becomes "KKVV". Indeed, this string doesn't have a substring "VK".

In the second sample, there are two optimal sequences of moves. One is "BVVKV"  →  "VBVKV"  →  "VVBKV". The other is "BVVKV"  →  "BVKVV"  →  "BKVVV".

In the fifth sample, no swaps are necessary.

C. Bear and Tree Jumps

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

A tree is an undirected connected graph without cycles. The distance between two vertices is the number of edges in a simple path between them.

Limak is a little polar bear. He lives in a tree that consists of *n* vertices, numbered 1 through *n*.

Limak recently learned how to jump. He can jump from a vertex to any vertex within distance at most *k*.

For a pair of vertices (*s*, *t*) we define *f*(*s*, *t*) as the minimum number of jumps Limak needs to get from *s* to *t*. Your task is to find the sum of *f*(*s*, *t*) over all pairs of vertices (*s*, *t*) such that *s* < *t*.

**Input**

The first line of the input contains two integers *n* and *k* (2 ≤ *n* ≤ 200 000, 1 ≤ *k* ≤ 5) — the number of vertices in the tree and the maximum allowed jump distance respectively.

The next *n* - 1 lines describe edges in the tree. The *i*-th of those lines contains two integers *ai* and *bi* (1 ≤ *ai*, *bi* ≤ *n*) — the indices on vertices connected with *i*-th edge.

It's guaranteed that the given edges form a tree.

**Output**

Print one integer, denoting the sum of *f*(*s*, *t*) over all pairs of vertices (*s*, *t*) such that *s* < *t*.

**Examples**

**input**

6 2  
1 2  
1 3  
2 4  
2 5  
4 6

**output**

20

**input**

13 3  
1 2  
3 2  
4 2  
5 2  
3 6  
10 6  
6 7  
6 13  
5 8  
5 9  
9 11  
11 12

**output**

114

**input**

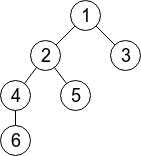
3 5  
2 1  
3 1

**output**

3

**Note**

In the first sample, the given tree has 6 vertices and it's displayed on the drawing below. Limak can jump to any vertex within distance at most 2. For example, from the vertex 5 he can jump to any of vertices: 1, 2 and 4 (well, he can also jump to the vertex 5 itself).



There are http://codeforces.com/predownloaded/8c/85/8c853bfcd2a10642e5c24fea21bf99142f851491.png pairs of vertices (*s*, *t*) such that *s* < *t*. For 5 of those pairs Limak would need two jumps: (1, 6), (3, 4), (3, 5), (3, 6), (5, 6). For other 10 pairs one jump is enough. So, the answer is 5·2 + 10·1 = 20.

In the third sample, Limak can jump between every two vertices directly. There are 3 pairs of vertices (*s* < *t*), so the answer is 3·1 = 3.

B. Bear and Different Names

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

In the army, it isn't easy to form a group of soldiers that will be effective on the battlefield. The communication is crucial and thus no two soldiers should share a name (what would happen if they got an order that Bob is a scouter, if there are two Bobs?).

A group of soldiers is effective if and only if their names are different. For example, a group (John, Bob, Limak) would be effective, while groups (Gary, Bob, Gary) and (Alice, Alice) wouldn't.

You are a spy in the enemy's camp. You noticed *n* soldiers standing in a row, numbered 1 through *n*. The general wants to choose a group of *k* consecutive soldiers. For every *k* consecutive soldiers, the general wrote down whether they would be an effective group or not.

You managed to steal the general's notes, with *n* - *k* + 1 strings *s*1, *s*2, ..., *sn*-*k*+ 1, each either "YES" or "NO".

* The string *s*1 describes a group of soldiers 1 through *k* ("YES" if the group is effective, and "NO" otherwise).
* The string *s*2 describes a group of soldiers 2 through *k* + 1.
* And so on, till the string *sn*-*k*+ 1 that describes a group of soldiers *n* - *k* + 1 through *n*.

Your task is to find possible names of *n* soldiers. Names should match the stolen notes. Each name should be a string that consists of between 1 and 10 English letters, inclusive. The first letter should be uppercase, and all other letters should be lowercase. Names don't have to be existing names — it's allowed to print "Xyzzzdj" or "T" for example.

Find and print any solution. It can be proved that there always exists at least one solution.

**Input**

The first line of the input contains two integers *n* and *k* (2 ≤ *k* ≤ *n* ≤ 50) — the number of soldiers and the size of a group respectively.

The second line contains *n* - *k* + 1 strings *s*1, *s*2, ..., *sn*-*k*+ 1. The string *si* is "YES" if the group of soldiers *i* through *i* + *k* - 1 is effective, and "NO" otherwise.

**Output**

Find any solution satisfying all given conditions. In one line print *n* space-separated strings, denoting possible names of soldiers in the order. The first letter of each name should be uppercase, while the other letters should be lowercase. Each name should contain English letters only and has length from 1 to 10.

If there are multiple valid solutions, print any of them.

**Examples**

**input**

8 3  
NO NO YES YES YES NO

**output**

Adam Bob Bob Cpqepqwer Limak Adam Bob Adam

**input**

9 8  
YES NO

**output**

R Q Ccccccccc Ccocc Ccc So Strong Samples Ccc

**input**

3 2  
NO NO

**output**

Na Na Na

**Note**

In the first sample, there are 8 soldiers. For every 3 consecutive ones we know whether they would be an effective group. Let's analyze the provided sample output:

* First three soldiers (i.e. Adam, Bob, Bob) wouldn't be an effective group because there are two Bobs. Indeed, the string *s*1 is "NO".
* Soldiers 2 through 4 (Bob, Bob, Cpqepqwer) wouldn't be effective either, and the string *s*2 is "NO".
* Soldiers 3 through 5 (Bob, Cpqepqwer, Limak) would be effective, and the string *s*3 is "YES".
* ...,
* Soldiers 6 through 8 (Adam, Bob, Adam) wouldn't be effective, and the string *s*6 is "NO".

A. Bear and Friendship Condition

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Bear Limak examines a social network. Its main functionality is that two members can become friends (then they can talk with each other and share funny pictures).

There are *n* members, numbered 1 through *n*. *m* pairs of members are friends. Of course, a member can't be a friend with themselves.

Let A-B denote that members A and B are friends. Limak thinks that a network is *reasonable* if and only if the following condition is satisfied: For every three **distinct** members (X, Y, Z), if X-Y and Y-Z then also X-Z.

For example: if Alan and Bob are friends, and Bob and Ciri are friends, then Alan and Ciri should be friends as well.

Can you help Limak and check if the network is reasonable? Print "YES" or "NO" accordingly, without the quotes.

**Input**

The first line of the input contain two integers *n* and *m* (3 ≤ *n* ≤ 150 000, http://codeforces.com/predownloaded/42/25/42252ce0112fbfd8e396b9ee524bb6cfe2cd93d4.png) — the number of members and the number of pairs of members that are friends.

The *i*-th of the next *m* lines contains two distinct integers *ai* and *bi* (1 ≤ *ai*, *bi* ≤ *n*, *ai* ≠ *bi*). Members *ai* and *bi* are friends with each other. No pair of members will appear more than once in the input.

**Output**

If the given network is reasonable, print "YES" in a single line (without the quotes). Otherwise, print "NO" in a single line (without the quotes).

**Examples**

**input**

4 3  
1 3  
3 4  
1 4

**output**

YES

**input**

4 4  
3 1  
2 3  
3 4  
1 2

**output**

NO

**input**

10 4  
4 3  
5 10  
8 9  
1 2

**output**

YES

**input**

3 2  
1 2  
2 3

**output**

NO

**Note**

The drawings below show the situation in the first sample (on the left) and in the second sample (on the right). Each edge represents two members that are friends. The answer is "NO" in the second sample because members (2, 3) are friends and members (3, 4) are friends, while members (2, 4) are not.



D. Draw Brackets!

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

A sequence of square brackets is regular if by inserting symbols "+" and "1" into it, you can get a regular mathematical expression from it. For example, sequences "[[]][]", "[]" and "[[][[]]]" — are regular, at the same time "][", "[[]" and "[[]]][" — are irregular.

Draw the given sequence using a minimalistic pseudographics in the strip of the lowest possible height — use symbols '+', '-' and '|'. For example, the sequence "[[][]][]" should be represented as:

+- -++- -+   
|+- -++- -+|| |  
|| || ||| |  
|+- -++- -+|| |  
+- -++- -+

Each bracket should be represented with the hepl of one or more symbols '|' (the vertical part) and symbols '+' and '-' as on the example which is given above.

Brackets should be drawn without spaces one by one, only dividing pairs of consecutive pairwise brackets with a single-space bar (so that the two brackets do not visually merge into one symbol). The image should have the minimum possible height.

The enclosed bracket is always smaller than the surrounding bracket, but each bracket separately strives to maximize the height of the image. So the pair of final brackets in the example above occupies the entire height of the image.

Study carefully the examples below, they adequately explain the condition of the problem. Pay attention that in this problem the answer (the image) is unique.

**Input**

The first line contains an even integer *n* (2 ≤ *n* ≤ 100) — the length of the sequence of brackets.

The second line contains the sequence of brackets — these are *n* symbols "[" and "]". It is guaranteed that the given sequence of brackets is regular.

**Output**

Print the drawn bracket sequence in the format which is given in the condition. Don't print extra (unnecessary) spaces.

**Examples**

**input**

8  
[[][]][]

**output**

+- -++- -+  
|+- -++- -+|| |  
|| || ||| |  
|+- -++- -+|| |  
+- -++- -+

**input**

6  
[[[]]]

**output**

+- -+  
|+- -+|  
||+- -+||  
||| |||  
||+- -+||  
|+- -+|  
+- -+

**input**

6  
[[][]]

**output**

+- -+  
|+- -++- -+|  
|| || ||  
|+- -++- -+|  
+- -+

**input**

2  
[]

**output**

+- -+  
| |  
+- -+

**input**

4  
[][]

**output**

+- -++- -+  
| || |  
+- -++- -+

C. Online Courses In BSU

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Now you can take online courses in the Berland State University! Polycarp needs to pass *k* **main** online courses of his specialty to get a diploma. In total *n* courses are availiable for the passage.

The situation is complicated by the dependence of online courses, for each course there is a list of those that must be passed before starting this online course (the list can be empty, it means that there is no limitation).

Help Polycarp to pass the least number of courses in total to get the specialty (it means to pass all **main** and necessary courses). Write a program which prints the order of courses.

Polycarp passes courses consistently, he starts the next course when he finishes the previous one. Each course can't be passed more than once.

**Input**

The first line contains *n* and *k* (1 ≤ *k* ≤ *n* ≤ 105) — the number of online-courses and the number of main courses of Polycarp's specialty.

The second line contains *k* distinct integers from 1 to *n* — numbers of main online-courses of Polycarp's specialty.

Then *n* lines follow, each of them describes the next course: the *i*-th of them corresponds to the course *i*. Each line starts from the integer *ti* (0 ≤ *ti* ≤ *n* - 1) — the number of courses on which the *i*-th depends. Then there follows the sequence of *ti* distinct integers from 1 to *n* — numbers of courses in random order, on which the *i*-th depends. It is guaranteed that no course can depend on itself.

It is guaranteed that the sum of all values *ti* doesn't exceed 105.

**Output**

Print -1, if there is no the way to get a specialty.

Otherwise, in the first line print the integer *m* — the minimum number of online-courses which it is necessary to pass to get a specialty. In the second line print *m* distinct integers — numbers of courses which it is necessary to pass in the chronological order of their passage. If there are several answers it is allowed to print any of them.

**Examples**

**input**

6 2  
5 3  
0  
0  
0  
2 2 1  
1 4  
1 5

**output**

5  
1 2 3 4 5

**input**

9 3  
3 9 5  
0  
0  
3 9 4 5  
0  
0  
1 8  
1 6  
1 2  
2 1 2

**output**

6  
1 2 9 4 5 3

**input**

3 3  
1 2 3  
1 2  
1 3  
1 1

**output**

-1

**Note**

In the first test firstly you can take courses number 1 and 2, after that you can take the course number 4, then you can take the course number 5, which is the main. After that you have to take only the course number 3, which is the last not passed main course.

B. Maximize Sum of Digits

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Anton has the integer *x*. He is interested what positive integer, which doesn't exceed *x*, has the maximum sum of digits.

Your task is to help Anton and to find the integer that interests him. If there are several such integers, determine the biggest of them.

**Input**

The first line contains the positive integer *x* (1 ≤ *x* ≤ 1018) — the integer which Anton has.

**Output**

Print the positive integer which doesn't exceed *x* and has the maximum sum of digits. If there are several such integers, print the biggest of them. Printed integer must not contain leading zeros.

**Examples**

**input**

100

**output**

99

**input**

48

**output**

48

**input**

521

**output**

499

A. New Password

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Innokentiy decides to change the password in the social net "Contact!", but he is too lazy to invent a new password by himself. That is why he needs your help.

Innokentiy decides that new password should satisfy the following conditions:

* the length of the password must be equal to *n*,
* the password should consist only of lowercase Latin letters,
* the number of distinct symbols in the password must be equal to *k*,
* any two consecutive symbols in the password must be distinct.

Your task is to help Innokentiy and to invent a new password which will satisfy all given conditions.

**Input**

The first line contains two positive integers *n* and *k* (2 ≤ *n* ≤ 100, 2 ≤ *k* ≤ *min*(*n*, 26)) — the length of the password and the number of distinct symbols in it.

Pay attention that a desired new password always exists.

**Output**

Print **any** password which satisfies all conditions given by Innokentiy.

**Examples**

**input**

4 3

**output**

java

**input**

6 6

**output**

python

**input**

5 2

**output**

phphp

**Note**

In the first test there is one of the appropriate new passwords — java, because its length is equal to 4 and 3 distinct lowercase letters a, j and v are used in it.

In the second test there is one of the appropriate new passwords — python, because its length is equal to 6 and it consists of 6 distinct lowercase letters.

In the third test there is one of the appropriate new passwords — phphp, because its length is equal to 5 and 2 distinct lowercase letters p and h are used in it.

Pay attention the condition that no two identical symbols are consecutive is correct for all appropriate passwords in tests.

D. k-Interesting Pairs Of Integers

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasya has the sequence consisting of *n* integers. Vasya consider the pair of integers *x* and *y* *k-interesting*, if their binary representation differs from each other exactly in *k* bits. For example, if *k* = 2, the pair of integers *x* = 5 and *y* = 3 is *k-interesting*, because their binary representation *x*=101 and *y*=011 differs exactly in two bits.

Vasya wants to know how many pairs of indexes (*i*, *j*) are in his sequence so that *i* < *j* and the pair of integers *ai* and *aj* is *k-interesting*. Your task is to help Vasya and determine this number.

**Input**

The first line contains two integers *n* and *k* (2 ≤ *n* ≤ 105, 0 ≤ *k* ≤ 14) — the number of integers in Vasya's sequence and the number of bits in which integers in *k-interesting* pair should differ.

The second line contains the sequence *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 104), which Vasya has.

**Output**

Print the number of pairs (*i*, *j*) so that *i* < *j* and the pair of integers *ai* and *aj* is *k-interesting*.

**Examples**

**input**

4 1  
0 3 2 1

**output**

4

**input**

6 0  
200 100 100 100 200 200

**output**

6

**Note**

In the first test there are 4 *k-interesting* pairs:

* (1, 3),
* (1, 4),
* (2, 3),
* (2, 4).

In the second test *k* = 0. Consequently, integers in any *k-interesting* pair should be equal to themselves. Thus, for the second test there are 6 *k-interesting* pairs:

* (1, 5),
* (1, 6),
* (2, 3),
* (2, 4),
* (3, 4),
* (5, 6).

C. Cycle In Maze

time limit per test

15 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

The Robot is in a rectangular maze of size *n* × *m*. Each cell of the maze is either empty or occupied by an obstacle. The Robot can move between neighboring cells on the side left (the symbol "L"), right (the symbol "R"), up (the symbol "U") or down (the symbol "D"). The Robot can move to the cell only if it is empty. Initially, the Robot is in the empty cell.

Your task is to find **lexicographically minimal** Robot's cycle with length **exactly** *k*, which begins and ends in the cell where the Robot was initially. It is allowed to the Robot to visit any cell many times (including starting).

Consider that Robot's way is given as a line which consists of symbols "L", "R", "U" and "D". For example, if firstly the Robot goes down, then left, then right and up, it means that his way is written as "DLRU".

In this task you **don't need** to minimize the length of the way. Find the minimum lexicographical (in alphabet order as in the dictionary) line which satisfies requirements above.

**Input**

The first line contains three integers *n*, *m* and *k* (1 ≤ *n*, *m* ≤ 1000, 1 ≤ *k* ≤ 106) — the size of the maze and the length of the cycle.

Each of the following *n* lines contains *m* symbols — the description of the maze. If the symbol equals to "." the current cell is empty. If the symbol equals to "\*" the current cell is occupied by an obstacle. If the symbol equals to "X" then initially the Robot is in this cell and it is empty. It is guaranteed that the symbol "X" is found in the maze exactly once.

**Output**

Print the lexicographically minimum Robot's way with the length exactly *k*, which starts and ends in the cell where initially Robot is. If there is no such way, print "IMPOSSIBLE"(without quotes).

**Examples**

**input**

2 3 2  
.\*\*  
X..

**output**

RL

**input**

5 6 14  
..\*\*\*.  
\*...X.  
..\*...  
..\*.\*\*  
....\*.

**output**

DLDDLLLRRRUURU

**input**

3 3 4  
\*\*\*  
\*X\*  
\*\*\*

**output**

IMPOSSIBLE

**Note**

In the first sample two cyclic ways for the Robot with the length 2 exist — "UD" and "RL". The second cycle is lexicographically less.

In the second sample the Robot should move in the following way: down, left, down, down, left, left, left, right, right, right, up, up, right, up.

In the third sample the Robot can't move to the neighboring cells, because they are occupied by obstacles.

B. News About Credit

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Polycarp studies at the university in the group which consists of *n* students (including himself). All they are registrated in the social net "TheContacnt!".

Not all students are equally sociable. About each student you know the value *ai* — the maximum number of messages which the *i*-th student is agree to send per day. The student can't send messages to himself.

In early morning Polycarp knew important news that the programming credit will be tomorrow. For this reason it is necessary to urgently inform all groupmates about this news using private messages.

Your task is to make a plan of using private messages, so that:

* the student *i* sends no more than *ai* messages (for all *i* from 1 to *n*);
* all students knew the news about the credit (initially only Polycarp knew it);
* the student can inform the other student only if he knows it himself.

Let's consider that all students are numerated by distinct numbers from 1 to *n*, and Polycarp **always** has the number 1.

In that task you shouldn't minimize the number of messages, the moment of time, when all knew about credit or some other parameters. Find any way how to use private messages which satisfies requirements above.

**Input**

The first line contains the positive integer *n* (2 ≤ *n* ≤ 100) — the number of students.

The second line contains the sequence *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 100), where *ai* equals to the maximum number of messages which can the *i*-th student agree to send. Consider that Polycarp **always** has the number 1.

**Output**

Print -1 to the first line if it is impossible to inform all students about credit.

Otherwise, in the first line print the integer *k* — the number of messages which will be sent. In each of the next *k* lines print two **distinct** integers *f* and *t*, meaning that the student number *f* sent the message with news to the student number *t*. All messages should be printed in chronological order. It means that the student, who is sending the message, must already know this news. It is assumed that students can receive repeated messages with news of the credit.

If there are several answers, it is acceptable to print any of them.

**Examples**

**input**

4  
1 2 1 0

**output**

3  
1 2  
2 4  
2 3

**input**

6  
2 0 1 3 2 0

**output**

6  
1 3  
3 4  
1 2  
4 5  
5 6  
4 6

**input**

3  
0 2 2

**output**

-1

**Note**

In the first test Polycarp (the student number 1) can send the message to the student number 2, who after that can send the message to students number 3 and 4. Thus, all students knew about the credit.

A. Year of University Entrance

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

There is the faculty of Computer Science in Berland. In the social net "TheContact!" for each course of this faculty there is the special group whose name equals the year of university entrance of corresponding course of students at the university.

Each of students joins the group of his course and joins all groups for which the year of student's university entrance differs by no more than *x* from the year of university entrance of this student, where *x* — some non-negative integer. A value *x* is not given, but it can be uniquely determined from the available data. Note that students don't join other groups.

You are given the list of groups which the student Igor joined. According to this information you need to determine the year of Igor's university entrance.

**Input**

The first line contains the positive odd integer *n* (1 ≤ *n* ≤ 5) — the number of groups which Igor joined.

The next line contains *n* distinct integers *a*1, *a*2, ..., *an* (2010 ≤ *ai* ≤ 2100) — years of student's university entrance for each group in which Igor is the member.

It is guaranteed that the input data is correct and the answer always exists. Groups are given randomly.

**Output**

Print the year of Igor's university entrance.

**Examples**

**input**

3  
2014 2016 2015

**output**

2015

**input**

1  
2050

**output**

2050

**Note**

In the first test the value *x* = 1. Igor entered the university in 2015. So he joined groups members of which are students who entered the university in 2014, 2015 and 2016.

In the second test the value *x* = 0. Igor entered only the group which corresponds to the year of his university entrance.

G. The Winds of Winter

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Given a rooted tree with *n* nodes. The Night King removes exactly one node from the tree and all the edges associated with it. Doing this splits the tree and forms a forest. The node which is removed is not a part of the forest.

The root of a tree in the forest is the node in that tree which does not have a parent. We define the strength of the forest as the size of largest tree in forest.

Jon Snow wants to minimize the strength of the forest. To do this he can perform the following operation at most once.

*He removes the edge between a node and its parent and inserts a new edge between this node and any other node in forest such that the total number of trees in forest remain same.*

For each node *v* you need to find the minimum value of strength of the forest formed when node *v* is removed.

**Input**

The first line of the input contains an integer *n* (1  ≤  *n*  ≤  105) — the number of vertices in the tree. Each of the next *n* lines contains a pair of vertex indices *ui* and *vi* (1  ≤  *ui*,  *vi*  ≤  *n*) where *ui* is the parent of *vi*. If *ui* = 0 then *vi* is the root.

**Output**

Print *n* line each containing a single integer. The *i*-th of them should be equal to minimum value of strength of forest formed when *i*-th node is removed and Jon Snow performs the operation described above at most once.

**Examples**

**input**

10  
0 1  
1 2  
1 3  
1 4  
2 5  
2 6  
3 7  
4 8  
4 9  
5 10

**output**

3  
4  
5  
5  
5  
9  
9  
9  
9  
9

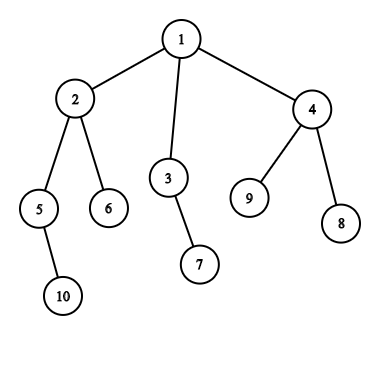
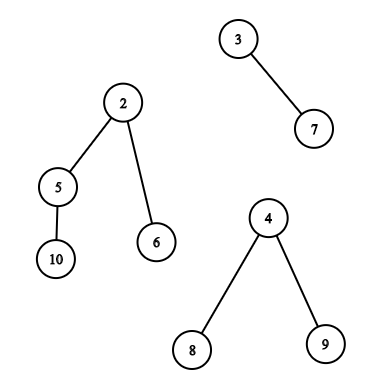
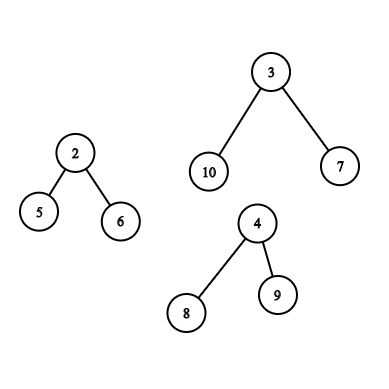
**input**

2  
2 1  
0 2

**output**

1  
1

**Note**

The tree for first test case is depicted below.When you remove the first node, the tree splits to form the following forest. The strength of this forest is 4.Jon Snow now changes the parent of vertex 10 from 5 to 3. The strength of forest now becomes 3.

F. Barrels and boxes

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Tarly has two different type of items, food boxes and wine barrels. There are *f* food boxes and *w* wine barrels. Tarly stores them in various stacks and each stack can consist of either food boxes or wine barrels but not both. The stacks are placed in a line such that no two stacks of food boxes are together and no two stacks of wine barrels are together.

The height of a stack is defined as the number of items in the stack. Two stacks are considered different if either their heights are different or one of them contains food and other contains wine.

Jon Snow doesn't like an arrangement if any stack of wine barrels has height less than or equal to *h*. What is the probability that Jon Snow will like the arrangement if all arrangement are equiprobably?

Two arrangement of stacks are considered different if exists such *i*, that *i*-th stack of one arrangement is different from the *i*-th stack of the other arrangement.

**Input**

The first line of input contains three integers *f*, *w*, *h* (0 ≤ *f*, *w*, *h* ≤ 105) — number of food boxes, number of wine barrels and *h* is as described above. It is guaranteed that he has at least one food box or at least one wine barrel.

**Output**

Output the probability that Jon Snow will like the arrangement. The probability is of the form http://codeforces.com/predownloaded/7b/21/7b213bbd1427432d309400301100bfaa85bd959f.png, then you need to output a single integer *p*·*q*- 1 *mod* (109 + 7).

**Examples**

**input**

1 1 1

**output**

0

**input**

1 2 1

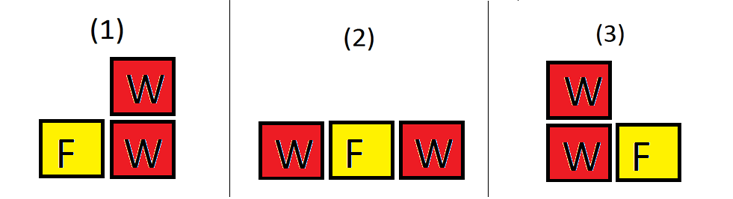
**output**

666666672

**Note**

In the first example *f*  =  1, *w* = 1 and *h* = 1, there are only two possible arrangement of stacks and Jon Snow doesn't like any of them.

In the second example *f* = 1, *w* = 2 and *h* = 1, there are three arrangements. Jon Snow likes the (1) and (3) arrangement. So the probabilty is http://codeforces.com/predownloaded/c4/38/c43839525c2cacb07e02a0d33be440ec0fa1dab3.png.



E. Game of Stones

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Sam has been teaching Jon the *Game of Stones* to sharpen his mind and help him devise a strategy to fight the white walkers. The rules of this game are quite simple:

* The game starts with *n* piles of stones indexed from 1 to *n*. The *i*-th pile contains *si* stones.
* The players make their moves alternatively. A move is considered as removal of some number of stones from a pile. Removal of 0stones does not count as a move.
* The player who is unable to make a move loses.

Now Jon believes that he is ready for battle, but Sam does not think so. To prove his argument, Sam suggested that they play a modified version of the game.

In this modified version, no move can be made more than once on a pile. For example, if 4 stones are removed from a pile, 4 stones cannot be removed from that pile again.

Sam sets up the game and makes the first move. Jon believes that Sam is just trying to prevent him from going to battle. Jon wants to know if he can win if both play optimally.

**Input**

First line consists of a single integer *n* (1 ≤ *n* ≤ 106) — the number of piles.

Each of next *n* lines contains an integer *si* (1 ≤ *si* ≤ 60) — the number of stones in *i*-th pile.

**Output**

Print a single line containing "YES" (without quotes) if Jon wins, otherwise print "NO" (without quotes)

**Examples**

**input**

1  
5

**output**

NO

**input**

2  
1  
2

**output**

YES

**Note**

In the first case, Sam removes all the stones and Jon loses.

In second case, the following moves are possible by Sam: http://codeforces.com/predownloaded/bd/c9/bdc96d549fa2f4c9b729f8387564021e6b94744c.png

In each of these cases, last move can be made by Jon to win the game as follows: http://codeforces.com/predownloaded/86/c0/86c065ddb4214f40f1ac2daaa173d807ad092ab6.png

D. Jon and Orbs

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Jon Snow is on the lookout for some orbs required to defeat the white walkers. There are *k* different types of orbs and he needs at least one of each. One orb spawns daily at the base of a Weirwood tree north of the wall. The probability of this orb being of any kind is equal. As the north of wall is full of dangers, he wants to know the minimum number of days he should wait before sending a ranger to collect the orbs such that the probability of him getting at least one of each kind of orb is at least http://codeforces.com/predownloaded/a6/62/a662db958644beef40900f5a41706492db8eea50.png, where ε < 10- 7.

To better prepare himself, he wants to know the answer for *q* different values of *pi*. Since he is busy designing the battle strategy with Sam, he asks you for your help.

**Input**

First line consists of two space separated integers *k*, *q* (1 ≤ *k*, *q* ≤ 1000) — number of different kinds of orbs and number of queries respectively.

Each of the next *q* lines contain a single integer *pi* (1 ≤ *pi* ≤ 1000) — *i*-th query.

**Output**

Output *q* lines. On *i*-th of them output single integer — answer for *i*-th query.

**Examples**

**input**

1 1  
1

**output**

1

**input**

2 2  
1  
2

**output**

2  
2

C. Jon Snow and his Favourite Number

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Jon Snow now has to fight with White Walkers. He has *n* rangers, each of which has his own strength. Also Jon Snow has his favourite number *x*. Each ranger can fight with a white walker only if the strength of the white walker equals his strength. He however thinks that his rangers are weak and need to improve. Jon now thinks that if he takes the bitwise XOR of strengths of some of rangers with his favourite number *x*, he might get soldiers of high strength. So, he decided to do the following operation *k* times:

1. Arrange all the rangers in a straight line in the order of increasing strengths.
2. Take the bitwise XOR (is written as http://codeforces.com/predownloaded/7b/ea/7beade55e90846d70020a3d03521d3458b66751b.png) of the strength of each alternate ranger with *x* and update it's strength.

Suppose, Jon has 5 rangers with strengths [9, 7, 11, 15, 5] and he performs the operation 1 time with *x* = 2. He first arranges them in the order of their strengths, [5, 7, 9, 11, 15]. Then he does the following:

1. The strength of first ranger is updated to http://codeforces.com/predownloaded/93/a9/93a90715158b04ab3500ffd59f5c44414479b984.png, i.e. 7.
2. The strength of second ranger remains the same, i.e. 7.
3. The strength of third ranger is updated to http://codeforces.com/predownloaded/7d/8a/7d8af2db200237c71088b93b52ceb33daef12c6a.png, i.e. 11.
4. The strength of fourth ranger remains the same, i.e. 11.
5. The strength of fifth ranger is updated to http://codeforces.com/predownloaded/98/6a/986af2c725ee421ce6d2a4ef8cce13ae92cb522f.png, i.e. 13.

The new strengths of the 5 rangers are [7, 7, 11, 11, 13]

Now, Jon wants to know the maximum and minimum strength of the rangers after performing the above operations *k* times. He wants your help for this task. Can you help him?

**Input**

First line consists of three integers *n*, *k*, *x* (1 ≤ *n* ≤ 105, 0 ≤ *k* ≤ 105, 0 ≤ *x* ≤ 103) — number of rangers Jon has, the number of times Jon will carry out the operation and Jon's favourite number respectively.

Second line consists of *n* integers representing the strengths of the rangers *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 103).

**Output**

Output two integers, the maximum and the minimum strength of the rangers after performing the operation *k* times.

**Examples**

**input**

5 1 2  
9 7 11 15 5

**output**

13 7

**input**

2 100000 569  
605 986

**output**

986 605

B. Code For 1

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Jon fought bravely to rescue the wildlings who were attacked by the white-walkers at Hardhome. On his arrival, Sam tells him that he wants to go to Oldtown to train at the Citadel to become a maester, so he can return and take the deceased Aemon's place as maester of Castle Black. Jon agrees to Sam's proposal and Sam sets off his journey to the Citadel. However becoming a trainee at the Citadel is not a cakewalk and hence the maesters at the Citadel gave Sam a problem to test his eligibility.

Initially Sam has a list with a single element *n*. Then he has to perform certain operations on this list. In each operation Sam must remove any element *x*, such that *x* > 1, from the list and insert at the same position http://codeforces.com/predownloaded/c8/fa/c8fa2489a03b7b1c83c08177c58cab5c307a4e1a.png, http://codeforces.com/predownloaded/56/34/563422a7e254284384497f49e8ef6a63ce4be486.png, http://codeforces.com/predownloaded/c8/fa/c8fa2489a03b7b1c83c08177c58cab5c307a4e1a.png sequentially. He must continue with these operations until all the elements in the list are either 0 or 1.

Now the masters want the total number of 1s in the range *l* to *r* (1-indexed). Sam wants to become a maester but unfortunately he cannot solve this problem. Can you help Sam to pass the eligibility test?

**Input**

The first line contains three integers *n*, *l*, *r* (0 ≤ *n* < 250, 0 ≤ *r* - *l* ≤ 105, *r* ≥ 1, *l* ≥ 1) – initial element and the range *l* to *r*.

It is guaranteed that *r* is not greater than the length of the final list.

**Output**

Output the total number of 1s in the range *l* to *r* in the final sequence.

**Examples**

**input**

7 2 5

**output**

4

**input**

10 3 10

**output**

5

**Note**

Consider first example:

http://codeforces.com/predownloaded/89/41/8941125368dcb6c235fd2256051a5127afa17479.png

Elements on positions from 2-nd to 5-th in list is [1, 1, 1, 1]. The number of ones is 4.

For the second example:

http://codeforces.com/predownloaded/5e/82/5e82ac21ce5cd1941b38aee2511565955d3bb35e.png

Elements on positions from 3-rd to 10-th in list is [1, 1, 1, 0, 1, 0, 1, 0]. The number of ones is 5.

A. Oath of the Night's Watch

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

*"Night gathers, and now my watch begins. It shall not end until my death. I shall take no wife, hold no lands, father no children. I shall wear no crowns and win no glory. I shall live and die at my post. I am the sword in the darkness. I am the watcher on the walls. I am the shield that guards the realms of men. I pledge my life and honor to the Night's Watch, for this night and all the nights to come."* — The Night's Watch oath.

With that begins the watch of Jon Snow. He is assigned the task to support the stewards.

This time he has *n* stewards with him whom he has to provide support. Each steward has his own strength. Jon Snow likes to support a steward only if there exists at least one steward who has strength strictly less than him and at least one steward who has strength strictly greater than him.

Can you find how many stewards will Jon support?

**Input**

First line consists of a single integer *n* (1 ≤ *n* ≤ 105) — the number of stewards with Jon Snow.

Second line consists of *n* space separated integers *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 109) representing the values assigned to the stewards.

**Output**

Output a single integer representing the number of stewards which Jon will feed.

**Examples**

**input**

2  
1 5

**output**

0

**input**

3  
1 2 5

**output**

1

**Note**

In the first sample, Jon Snow cannot support steward with strength 1 because there is no steward with strength less than 1 and he cannot support steward with strength 5 because there is no steward with strength greater than 5.

In the second sample, Jon Snow can support steward with strength 2 because there are stewards with strength less than 2 and greater than 2.

E. Change-free

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Student Arseny likes to plan his life for *n* days ahead. He visits a canteen every day and he has already decided what he will order in each of the following *n* days. Prices in the canteen do not change and that means Arseny will spend *ci* rubles during the *i*-th day.

There are 1-ruble coins and 100-ruble notes in circulation. At this moment, Arseny has *m* coins and a sufficiently large amount of notes (you can assume that he has an infinite amount of them). Arseny loves modern technologies, so he uses his credit card everywhere except the canteen, but he has to pay in cash in the canteen because it does not accept cards.

Cashier always asks the student to pay change-free. However, it's not always possible, but Arseny tries to minimize the *dissatisfaction* of the cashier. Cashier's dissatisfaction for each of the days is determined by the total amount of notes and coins in the change. To be precise, if the cashier gives Arseny *x* notes and coins on the *i*-th day, his dissatisfaction for this day equals *x*·*wi*. Cashier always gives change using as little coins and notes as possible, he always has enough of them to be able to do this.

"Caution! Angry cashier"

Arseny wants to pay in such a way that the total dissatisfaction of the cashier for *n* days would be as small as possible. Help him to find out how he needs to pay in each of the *n* days!

Note that Arseny always has enough money to pay, because he has an infinite amount of notes. Arseny can use notes and coins he received in change during any of the following days.

**Input**

The first line contains two integers *n* and *m* (1 ≤ *n* ≤ 105, 0 ≤ *m* ≤ 109) — the amount of days Arseny planned his actions for and the amount of coins he currently has.

The second line contains a sequence of integers *c*1, *c*2, ..., *cn* (1 ≤ *ci* ≤ 105) — the amounts of money in rubles which Arseny is going to spend for each of the following days.

The third line contains a sequence of integers *w*1, *w*2, ..., *wn* (1 ≤ *wi* ≤ 105) — the cashier's dissatisfaction coefficients for each of the following days.

**Output**

In the first line print one integer — minimum possible total dissatisfaction of the cashier.

Then print *n* lines, the *i*-th of then should contain two numbers — the amount of notes and the amount of coins which Arseny should use to pay in the canteen on the *i*-th day.

Of course, the total amount of money Arseny gives to the casher in any of the days should be no less than the amount of money he has planned to spend. It also shouldn't exceed 106 rubles: Arseny never carries large sums of money with him.

If there are multiple answers, print any of them.

**Examples**

**input**

5 42  
117 71 150 243 200  
1 1 1 1 1

**output**

79  
1 17  
1 0  
2 0  
2 43  
2 0

**input**

3 0  
100 50 50  
1 3 2

**output**

150  
1 0  
1 0  
0 50

**input**

5 42  
117 71 150 243 200  
5 4 3 2 1

**output**

230  
1 17  
1 0  
1 50  
3 0  
2 0

D. Cartons of milk

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Olya likes milk very much. She drinks *k* cartons of milk each day if she has at least *k* and drinks all of them if she doesn't. But there's an issue — expiration dates. Each carton has a date after which you can't drink it (you still can drink it exactly at the date written on the carton). Due to this, if Olya's fridge contains a carton past its expiry date, she throws it away.

Olya hates throwing out cartons, so when she drinks a carton, she chooses the one which expires the fastest. It's easy to understand that this strategy minimizes the amount of cartons thrown out and lets her avoid it if it's even possible.

Milk. Best before: 20.02.2017.

The main issue Olya has is the one of buying new cartons. Currently, there are *n* cartons of milk in Olya's fridge, for each one an expiration date is known (how soon does it expire, measured in days). In the shop that Olya visited there are *m* cartons, and the expiration date is known for each of those cartons as well.

Find the maximum number of cartons Olya can buy so that she wouldn't have to throw away any cartons. Assume that Olya drank no cartons today.

**Input**

In the first line there are three integers *n*, *m*, *k* (1 ≤ *n*, *m* ≤ 106, 1 ≤ *k* ≤ *n* + *m*) — the amount of cartons in Olya's fridge, the amount of cartons in the shop and the number of cartons Olya drinks each day.

In the second line there are *n* integers *f*1, *f*2, ..., *fn* (0 ≤ *fi* ≤ 107) — expiration dates of the cartons in Olya's fridge. The expiration date is expressed by the number of days the drinking of this carton can be delayed. For example, a 0 expiration date means it must be drunk today, 1 — no later than tomorrow, etc.

In the third line there are *m* integers *s*1, *s*2, ..., *sm* (0 ≤ *si* ≤ 107) — expiration dates of the cartons in the shop in a similar format.

**Output**

If there's no way for Olya to drink the cartons she already has in her fridge, print -1.

Otherwise, in the first line print the maximum number *x* of cartons which Olya can buy so that she wouldn't have to throw a carton away. The next line should contain exactly *x* integers — the numbers of the cartons that should be bought (cartons are numbered in an order in which they are written in the input, starting with 1). Numbers should not repeat, but can be in arbitrary order. If there are multiple correct answers, print any of them.

**Examples**

**input**

3 6 2  
1 0 1  
2 0 2 0 0 2

**output**

3  
1 2 3

**input**

3 1 2  
0 0 0  
1

**output**

-1

**input**

2 1 2  
0 1  
0

**output**

1  
1

**Note**

In the first example *k* = 2 and Olya has three cartons with expiry dates 0, 1 and 1 (they expire today, tomorrow and tomorrow), and the shop has 3 cartons with expiry date 0 and 3 cartons with expiry date 2. Olya can buy three cartons, for example, one with the expiry date 0 and two with expiry date 2.

In the second example all three cartons Olya owns expire today and it means she would have to throw packets away regardless of whether she buys an extra one or not.

In the third example Olya would drink *k* = 2 cartons today (one she alreay has in her fridge and one from the shop) and the remaining one tomorrow.

1. Garland

767C

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Once at New Year Dima had a dream in which he was presented a fairy garland. A garland is a set of lamps, some pairs of which are connected by wires. Dima remembered that each two lamps in the garland were connected directly or indirectly via some wires. Furthermore, the number of wires was exactly one less than the number of lamps.

There was something unusual about the garland. Each lamp had its own brightness which depended on the temperature of the lamp. Temperatures could be positive, negative or zero. Dima has two friends, so he decided to share the garland with them. He wants to cut two different wires so that the garland breaks up into three parts. Each part of the garland should shine equally, i. e. the sums of lamps' temperatures should be equal in each of the parts. Of course, each of the parts should be non-empty, i. e. each part should contain at least one lamp.

Help Dima to find a suitable way to cut the garland, or determine that this is impossible.

While examining the garland, Dima lifted it up holding by one of the lamps. Thus, each of the lamps, except the one he is holding by, is now hanging on some wire. So, you should print two lamp ids as the answer which denote that Dima should cut the wires these lamps are hanging on. Of course, the lamp Dima is holding the garland by can't be included in the answer.

**Input**

The first line contains single integer *n* (3 ≤ *n* ≤ 106) — the number of lamps in the garland.

Then *n* lines follow. The *i*-th of them contain the information about the *i*-th lamp: the number lamp *ai*, it is hanging on (and 0, if is there is no such lamp), and its temperature *ti* ( - 100 ≤ *ti* ≤ 100). The lamps are numbered from 1 to *n*.

**Output**

If there is no solution, print -1.

Otherwise print two integers — the indexes of the lamps which mean Dima should cut the wires they are hanging on. If there are multiple answers, print any of them.

**Examples**

**input**

6  
2 4  
0 5  
4 2  
2 1  
1 1  
4 2

**output**

1 4

**input**

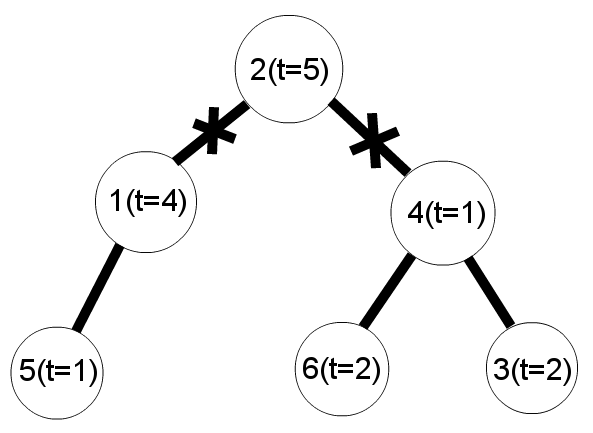
6  
2 4  
0 6  
4 2  
2 1  
1 1  
4 2

**output**

-1

**Note**

The garland and cuts scheme for the first example:



B. The Queue

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Finally! Vasya have come of age and that means he can finally get a passport! To do it, he needs to visit the passport office, but it's not that simple. There's only one receptionist at the passport office and people can queue up long before it actually opens. Vasya wants to visit the passport office tomorrow.

He knows that the receptionist starts working after *ts* minutes have passed after midnight and closes after *tf* minutes have passed after midnight (so that (*tf* - 1) is the last minute when the receptionist is still working). The receptionist spends exactly *t* minutes on each person in the queue. If the receptionist would stop working within *t* minutes, he stops serving visitors (other than the one he already serves).

Vasya also knows that exactly *n* visitors would come tomorrow. For each visitor Vasya knows the point of time when he would come to the passport office. Each visitor queues up and doesn't leave until he was served. If the receptionist is free when a visitor comes (in particular, if the previous visitor was just served and the queue is empty), the receptionist begins to serve the newcomer immediately.

"Reception 1"

For each visitor, the point of time when he would come to the passport office is positive. Vasya can come to the office at the time zero (that is, at midnight) if he needs so, but he can come to the office only at integer points of time. If Vasya arrives at the passport office at the same time with several other visitors, he yields to them and stand in the queue after the last of them.

Vasya wants to come at such point of time that he will be served by the receptionist, and he would spend the minimum possible time in the queue. Help him!

**Input**

The first line contains three integers: the point of time when the receptionist begins to work *ts*, the point of time when the receptionist stops working *tf* and the time the receptionist spends on each visitor *t*. The second line contains one integer *n* — the amount of visitors (0 ≤ *n* ≤ 100 000). The third line contains positive integers in non-decreasing order — the points of time when the visitors arrive to the passport office.

All times are set in minutes and do not exceed 1012; it is guaranteed that *ts* < *tf*. It is also guaranteed that Vasya can arrive at the passport office at such a point of time that he would be served by the receptionist.

**Output**

Print single non-negative integer — the point of time when Vasya should arrive at the passport office. If Vasya arrives at the passport office at the same time with several other visitors, he yields to them and queues up the last. If there are many answers, you can print any of them.

**Examples**

**input**

10 15 2  
2  
10 13

**output**

12

**input**

8 17 3  
4  
3 4 5 8

**output**

2

**Note**

In the first example the first visitor comes exactly at the point of time when the receptionist begins to work, and he is served for two minutes. At 12 minutes after the midnight the receptionist stops serving the first visitor, and if Vasya arrives at this moment, he will be served immediately, because the next visitor would only come at 13 minutes after midnight.

In the second example, Vasya has to come before anyone else to be served.

A. Snacktower

time limit per test

2 seconds

memory limit per test

256 megabytes

input

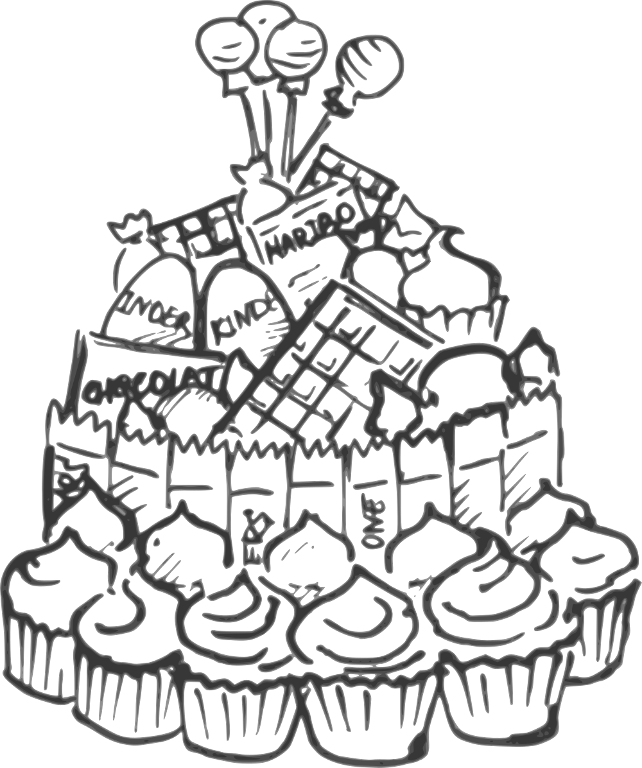
standard input

output

standard output

According to an old legeng, a long time ago Ankh-Morpork residents did something wrong to miss Fortune, and she cursed them. She said that at some time *n* snacks of distinct sizes will fall on the city, and the residents should build a Snacktower of them by placing snacks one on another. Of course, big snacks should be at the bottom of the tower, while small snacks should be at the top.

Years passed, and once different snacks started to fall onto the city, and the residents began to build the Snacktower.



However, they faced some troubles. Each day exactly one snack fell onto the city, but their order was strange. So, at some days the residents weren't able to put the new stack on the top of the Snacktower: they had to wait until all the bigger snacks fell. Of course, in order to not to anger miss Fortune again, the residents placed each snack on the top of the tower immediately as they could do it.

Write a program that models the behavior of Ankh-Morpork residents.

**Input**

The first line contains single integer *n* (1 ≤ *n* ≤ 100 000) — the total number of snacks.

The second line contains *n* integers, the *i*-th of them equals the size of the snack which fell on the *i*-th day. Sizes are distinct integers from 1 to *n*.

**Output**

Print *n* lines. On the *i*-th of them print the sizes of the snacks which the residents placed on the top of the Snacktower on the *i*-th day in the order they will do that. If no snack is placed on some day, leave the corresponding line empty.

**Examples**

**input**

3  
3 1 2

**output**

3  
   
2 1

**input**

5  
4 5 1 2 3

**output**

5 4  
   
   
3 2 1

**Note**

In the example a snack of size 3 fell on the first day, and the residents immediately placed it. On the second day a snack of size 1 fell, and the residents weren't able to place it because they were missing the snack of size 2. On the third day a snack of size 2 fell, and the residents immediately placed it. Right after that they placed the snack of size 1 which had fallen before.

E. Mahmoud and a xor trip

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Mahmoud and Ehab live in a country with *n* cities numbered from 1 to *n* and connected by *n* - 1 undirected roads. It's guaranteed that you can reach any city from any other using these roads. Each city has a number *ai* attached to it.

We define the distance from city *x* to city *y* as the xor of numbers attached to the cities on the path from *x* to *y* **(including both *x* and *y*)**. In other words if values attached to the cities on the path from *x* to *y* form an array *p* of length *l* then the distance between them is http://codeforces.com/predownloaded/94/30/9430dbaf9626a574a6d7f5a8daac1a8584f82c6e.png, where http://codeforces.com/predownloaded/7b/ea/7beade55e90846d70020a3d03521d3458b66751b.png is bitwise xor operation.

Mahmoud and Ehab want to choose two cities and make a journey from one to another. The index of the start city is always less than or equal to the index of the finish city (they may start and finish in the same city and in this case the distance equals the number attached to that city). They can't determine the two cities so they try every city as a start and every city with greater index as a finish. They want to know the total distance between all pairs of cities.

**Input**

The first line contains integer *n* (1 ≤ *n* ≤ 105) — the number of cities in Mahmoud and Ehab's country.

Then the second line contains *n* integers *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 106) which represent the numbers attached to the cities. Integer *ai* is attached to the city *i*.

Each of the next *n*  -  1 lines contains two integers *u* and *v* (1  ≤  *u*,  *v*  ≤  *n*, *u*  ≠  *v*), denoting that there is an undirected road between cities *u* and *v*. It's guaranteed that you can reach any city from any other using these roads.

**Output**

Output one number denoting the total distance between all pairs of cities.

**Examples**

**input**

3  
1 2 3  
1 2  
2 3

**output**

10

**input**

5  
1 2 3 4 5  
1 2  
2 3  
3 4  
3 5

**output**

52

**input**

5  
10 9 8 7 6  
1 2  
2 3  
3 4  
3 5

**output**

131

**Note**

A bitwise xor takes two bit integers of equal length and performs the logical xor operation on each pair of corresponding bits. The result in each position is 1 if only the first bit is 1 or only the second bit is 1, but will be 0 if both are 0 or both are 1. You can read more about bitwise xor operation here: <https://en.wikipedia.org/wiki/Bitwise_operation#XOR>.

In the first sample the available paths are:

* city 1 to itself with a distance of 1,
* city 2 to itself with a distance of 2,
* city 3 to itself with a distance of 3,
* city 1 to city 2 with a distance of http://codeforces.com/predownloaded/f3/f4/f3f48ca9259e359e0f860214756305ac8291888b.png,
* city 1 to city 3 with a distance of http://codeforces.com/predownloaded/ab/15/ab159d27df504d5fa70b458bf50f0ca291806210.png,
* city 2 to city 3 with a distance of http://codeforces.com/predownloaded/c5/67/c5679f470aa199c954d8f147670732be255009b5.png.

The total distance between all pairs of cities equals 1 + 2 + 3 + 3 + 0 + 1 = 10.

D. Mahmoud and a Dictionary

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Mahmoud wants to write a new dictionary that contains *n* words and relations between them. There are two types of relations: synonymy (i. e. the two words mean the same) and antonymy (i. e. the two words mean the opposite). From time to time he discovers a new relation between two words.

He know that if two words have a relation between them, then each of them has relations with the words that has relations with the other. For example, if like means love and love is the opposite of hate, then like is also the opposite of hate. One more example: if love is the opposite of hate and hate is the opposite of like, then love means like, and so on.

Sometimes Mahmoud discovers a wrong relation. A wrong relation is a relation that makes two words equal and opposite at the same time. For example if he knows that love means like and like is the opposite of hate, and then he figures out that hate means like, the last relation is absolutely wrong because it makes hate and like opposite and have the same meaning at the same time.

After Mahmoud figured out many relations, he was worried that some of them were wrong so that they will make other relations also wrong, so he decided to tell every relation he figured out to his coder friend Ehab and for every relation he wanted to know is it correct or wrong, basing on the previously discovered relations. If it is wrong he ignores it, and doesn't check with following relations.

After adding all relations, Mahmoud asked Ehab about relations between some words based on the information he had given to him. Ehab is busy making a Codeforces round so he asked you for help.

**Input**

The first line of input contains three integers *n*, *m* and *q* (2 ≤ *n* ≤ 105, 1 ≤ *m*, *q* ≤ 105) where *n* is the number of words in the dictionary, *m* is the number of relations Mahmoud figured out and *q* is the number of questions Mahmoud asked after telling all relations.

The second line contains *n* distinct words *a*1, *a*2, ..., *an* consisting of small English letters with length not exceeding 20, which are the words in the dictionary.

Then *m* lines follow, each of them contains an integer *t* (1 ≤ *t* ≤ 2) followed by two different words *xi* and *yi* which has appeared in the dictionary words. If *t* = 1, that means *xi* has a synonymy relation with *yi*, otherwise *xi* has an antonymy relation with *yi*.

Then *q* lines follow, each of them contains two different words which has appeared in the dictionary. That are the pairs of words Mahmoud wants to know the relation between basing on the relations he had discovered.

All words in input contain only lowercase English letters and their lengths don't exceed 20 characters. In all relations and in all questions the two words are different.

**Output**

First, print *m* lines, one per each relation. If some relation is wrong (makes two words opposite and have the same meaning at the same time) you should print "NO" (without quotes) and ignore it, otherwise print "YES" (without quotes).

After that print *q* lines, one per each question. If the two words have the same meaning, output 1. If they are opposites, output 2. If there is no relation between them, output 3.

See the samples for better understanding.

**Examples**

**input**

3 3 4  
hate love like  
1 love like  
2 love hate  
1 hate like  
love like  
love hate  
like hate  
hate like

**output**

YES  
YES  
NO  
1  
2  
2  
2

**input**

8 6 5  
hi welcome hello ihateyou goaway dog cat rat  
1 hi welcome  
1 ihateyou goaway  
2 hello ihateyou  
2 hi goaway  
2 hi hello  
1 hi hello  
dog cat  
dog hi  
hi hello  
ihateyou goaway  
welcome ihateyou

**output**

YES  
YES  
YES  
YES  
NO  
YES  
3  
3  
1  
1  
2

C. Mahmoud and a Message

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Mahmoud wrote a message *s* of length *n*. He wants to send it as a birthday present to his friend Moaz who likes strings. He wrote it on a magical paper but he was surprised because some characters disappeared while writing the string. That's because this magical paper doesn't allow character number *i* in the English alphabet to be written on it in a string of length more than *ai*. For example, if *a*1 = 2 he can't write character 'a' on this paper in a string of length 3 or more. String "aa" is allowed while string "aaa" is not.

Mahmoud decided to split the message into some non-empty substrings so that he can write every substring on an independent magical paper and fulfill the condition. The sum of their lengths should be *n* and they shouldn't overlap. For example, if *a*1 = 2 and he wants to send string "aaa", he can split it into "a" and "aa" and use 2 magical papers, or into "a", "a" and "a" and use 3 magical papers. He can't split it into "aa" and "aa" because the sum of their lengths is greater than *n*. He can split the message into single string if it fulfills the conditions.

A substring of string *s* is a string that consists of some consecutive characters from string *s*, strings "ab", "abc" and "b" are substrings of string "abc", while strings "acb" and "ac" are not. Any string is a substring of itself.

While Mahmoud was thinking of how to split the message, Ehab told him that there are many ways to split it. After that Mahmoud asked you three questions:

* How many ways are there to split the string into substrings such that every substring fulfills the condition of the magical paper, the sum of their lengths is *n* and they don't overlap? Compute the answer modulo 109 + 7.
* What is the maximum length of a substring that can appear in some valid splitting?
* What is the minimum number of substrings the message can be spit in?

Two ways are considered different, if the sets of split positions differ. For example, splitting "aa|a" and "a|aa" are considered different splittings of message "aaa".

**Input**

The first line contains an integer *n* (1 ≤ *n* ≤ 103) denoting the length of the message.

The second line contains the message *s* of length *n* that consists of lowercase English letters.

The third line contains 26 integers *a*1, *a*2, ..., *a*26 (1 ≤ *ax* ≤ 103) — the maximum lengths of substring each letter can appear in.

**Output**

Print three lines.

In the first line print the number of ways to split the message into substrings and fulfill the conditions mentioned in the problem modulo 109  +  7.

In the second line print the length of the longest substring over all the ways.

In the third line print the minimum number of substrings over all the ways.

**Examples**

**input**

3  
aab  
2 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

**output**

3  
2  
2

**input**

10  
abcdeabcde  
5 5 5 5 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

**output**

401  
4  
3

**Note**

In the first example the three ways to split the message are:

* a|a|b
* aa|b
* a|ab

The longest substrings are "aa" and "ab" of length 2.

The minimum number of substrings is 2 in "a|ab" or "aa|b".

Notice that "aab" is not a possible splitting because the letter 'a' appears in a substring of length 3, while *a*1 = 2.

B. Mahmoud and a Triangle

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Mahmoud has *n* line segments, the *i*-th of them has length *ai*. Ehab challenged him to use **exactly 3** line segments to form a non-degenerate triangle. Mahmoud doesn't accept challenges unless he is sure he can win, so he asked you to tell him if he should accept the challenge. Given the lengths of the line segments, check if he can choose exactly 3 of them to form a non-degenerate triangle.

Mahmoud should use exactly 3 line segments, he can't concatenate two line segments or change any length. A non-degenerate triangle is a triangle with positive area.

**Input**

The first line contains single integer *n* (3 ≤ *n* ≤ 105) — the number of line segments Mahmoud has.

The second line contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109) — the lengths of line segments Mahmoud has.

**Output**

In the only line print "YES" if he can choose exactly three line segments and form a non-degenerate triangle with them, and "NO" otherwise.

**Examples**

**input**

5  
1 5 3 2 4

**output**

YES

**input**

3  
4 1 2

**output**

NO

**Note**

For the first example, he can use line segments with lengths 2, 4 and 5 to form a non-degenerate triangle.

A. Mahmoud and Longest Uncommon Subsequence

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

While Mahmoud and Ehab were practicing for IOI, they found a problem which name was Longest common subsequence. They solved it, and then Ehab challenged Mahmoud with another problem.

Given two strings *a* and *b*, find the length of their longest uncommon subsequence, which is the longest string that is a subsequence of one of them and not a subsequence of the other.

A subsequence of some string is a sequence of characters that appears in the same order in the string, The appearances don't have to be consecutive, for example, strings "ac", "bc", "abc" and "a" are subsequences of string "abc" while strings "abbc" and "acb" are not. The empty string is a subsequence of any string. Any string is a subsequence of itself.

**Input**

The first line contains string *a*, and the second line — string *b*. Both of these strings are non-empty and consist of lowercase letters of English alphabet. The length of each string is not bigger than 105 characters.

**Output**

If there's no uncommon subsequence, print "-1". Otherwise print the length of the longest uncommon subsequence of *a* and *b*.

**Examples**

**input**

abcd  
defgh

**output**

5

**input**

a  
a

**output**

-1

**Note**

In the first example: you can choose "defgh" from string *b* as it is the longest subsequence of string *b* that doesn't appear as a subsequence of string *a*.

G. Math, math everywhere

time limit per test

5 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

*If you have gone that far, you'll probably skip unnecessary legends anyway...*

You are given a binary string http://codeforces.com/predownloaded/b1/61/b161c0248819ce2267d98c519d3f7e187cb60ef8.png and an integer http://codeforces.com/predownloaded/8c/e6/8ce6aea54df8954cf29df3d9aaad819e1abfa5fb.png. Find the number of integers *k*, 0 ≤ *k* < *N*, such that for all *i* = 0, 1, ..., *m* - 1

http://codeforces.com/predownloaded/56/00/560020e71ab9fe8e78721bb258574821af5d03c3.png

Print the answer modulo 109 + 7.

**Input**

In the first line of input there is a string *s* consisting of 0's and 1's (1 ≤ |*s*| ≤ 40).

In the next line of input there is an integer *n* (1 ≤ *n* ≤ 5·105).

Each of the next *n* lines contains two space-separated integers *pi*, α*i* (1 ≤ *pi*, α*i* ≤ 109, *pi* is prime). All *pi* are distinct.

**Output**

A single integer — the answer to the problem.

**Examples**

**input**

1  
2  
2 1  
3 1

**output**

2

**input**

01  
2  
3 2  
5 1

**output**

15

**input**

1011  
1  
3 1000000000

**output**

411979884

F. Souvenirs

time limit per test

3 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Artsem is on vacation and wants to buy souvenirs for his two teammates. There are *n* souvenir shops along the street. In *i*-th shop Artsem can buy one souvenir for *ai* dollars, and he cannot buy more than one souvenir in one shop. He doesn't want to introduce envy in his team, so he wants to buy two souvenirs with least possible difference in price.

Artsem has visited the shopping street *m* times. For some strange reason on the *i*-th day only shops with numbers from *li* to *ri* were operating (weird? yes it is, but have you ever tried to come up with a reasonable legend for a range query problem?). For each visit, Artsem wants to know the minimum possible difference in prices of two different souvenirs he can buy in the opened shops.

In other words, for each Artsem's visit you should find the minimum possible value of |*as* - *at*| where *li* ≤ *s*, *t* ≤ *ri*, *s* ≠ *t*.

**Input**

The first line contains an integer *n* (2 ≤ *n* ≤ 105).

The second line contains *n* space-separated integers *a*1, ..., *an* (0 ≤ *ai* ≤ 109).

The third line contains the number of queries *m* (1 ≤ *m* ≤ 3·105).

Next *m* lines describe the queries. *i*-th of these lines contains two space-separated integers *li* and *ri* denoting the range of shops working on *i*-th day (1 ≤ *li* < *ri* ≤ *n*).

**Output**

Print the answer to each query in a separate line.

**Example**

**input**

8  
3 1 4 1 5 9 2 6  
4  
1 8  
1 3  
4 8  
5 7

**output**

0  
1  
1  
3

E. Tree Folding

time limit per test

2 seconds

memory limit per test

512 megabytes

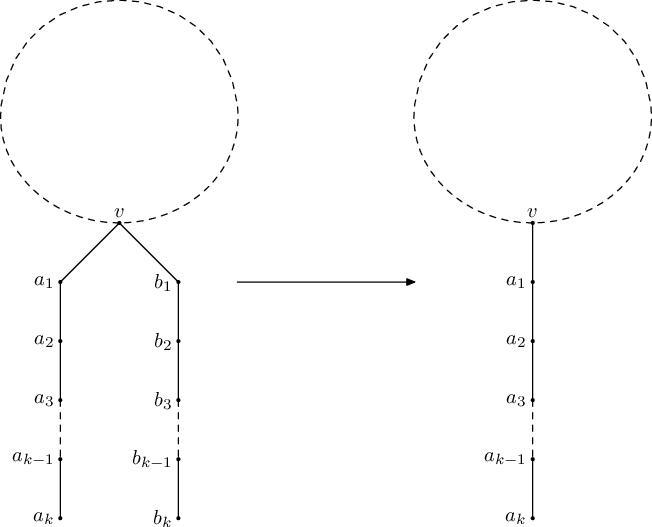
input

standard input

output

standard output

Vanya wants to minimize a tree. He can perform the following operation multiple times: choose a vertex *v*, and two disjoint (except for *v*) paths of equal length *a*0 = *v*, *a*1, ..., *ak*, and *b*0 = *v*, *b*1, ..., *bk*. Additionally, vertices *a*1, ..., *ak*, *b*1, ..., *bk* must not have any neighbours in the tree other than adjacent vertices of corresponding paths. After that, one of the paths may be merged into the other, that is, the vertices *b*1, ..., *bk* can be effectively erased:



Help Vanya determine if it possible to make the tree into a path via a sequence of described operations, and if the answer is positive, also determine the shortest length of such path.

**Input**

The first line of input contains the number of vertices *n* (2 ≤ *n* ≤ 2·105).

Next *n* - 1 lines describe edges of the tree. Each of these lines contains two space-separated integers *u* and *v* (1 ≤ *u*, *v* ≤ *n*, *u* ≠ *v*) — indices of endpoints of the corresponding edge. It is guaranteed that the given graph is a tree.

**Output**

If it is impossible to obtain a path, print -1. Otherwise, print the minimum number of edges in a possible path.

**Examples**

**input**

6  
1 2  
2 3  
2 4  
4 5  
1 6

**output**

3

**input**

7  
1 2  
1 3  
3 4  
1 5  
5 6  
6 7

**output**

-1

**Note**

In the first sample case, a path of three edges is obtained after merging paths 2 - 1 - 6 and 2 - 4 - 5.

It is impossible to perform any operation in the second sample case. For example, it is impossible to merge paths 1 - 3 - 4 and 1 - 5 - 6, since vertex 6 additionally has a neighbour 7 that is not present in the corresponding path.

D. Artsem and Saunders

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Artsem has a friend Saunders from University of Chicago. Saunders presented him with the following problem.

Let [*n*] denote the set {1, ..., *n*}. We will also write *f*: [*x*] → [*y*] when a function *f* is defined in integer points 1, ..., *x*, and all its values are integers from 1 to *y*.

Now then, you are given a function *f*: [*n*] → [*n*]. Your task is to find a positive integer *m*, and two functions *g*: [*n*] → [*m*], *h*: [*m*] → [*n*], such that *g*(*h*(*x*)) = *x* for all http://codeforces.com/predownloaded/e8/df/e8df9cdcdcdd9dc483cfffa96e3dfba73bd4911b.png, and *h*(*g*(*x*)) = *f*(*x*) for all http://codeforces.com/predownloaded/6c/71/6c71d5f0e27576f1e962d0fbae5dce915abbab38.png, or determine that finding these is impossible.

**Input**

The first line contains an integer *n* (1 ≤ *n* ≤ 105).

The second line contains *n* space-separated integers — values *f*(1), ..., *f*(*n*) (1 ≤ *f*(*i*) ≤ *n*).

**Output**

If there is no answer, print one integer -1.

Otherwise, on the first line print the number *m* (1 ≤ *m* ≤ 106). On the second line print *n* numbers *g*(1), ..., *g*(*n*). On the third line print *m* numbers *h*(1), ..., *h*(*m*).

If there are several correct answers, you may output any of them. It is guaranteed that if a valid answer exists, then there is an answer satisfying the above restrictions.

**Examples**

**input**

3  
1 2 3

**output**

3  
1 2 3  
1 2 3

**input**

3  
2 2 2

**output**

1  
1 1 1  
2

**input**

2  
2 1

**output**

-1

C. Table Tennis Game 2

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Misha and Vanya have played several table tennis sets. Each set consists of several serves, each serve is won by one of the players, he receives one point and the loser receives nothing. Once one of the players scores exactly *k* points, the score is reset and a new set begins.

Across all the sets Misha scored *a* points in total, and Vanya scored *b* points. Given this information, determine the maximum number of sets they could have played, or that the situation is impossible.

Note that the game consisted of several complete sets.

**Input**

The first line contains three space-separated integers *k*, *a* and *b* (1 ≤ *k* ≤ 109, 0 ≤ *a*, *b* ≤ 109, *a* + *b* > 0).

**Output**

If the situation is impossible, print a single number -1. Otherwise, print the maximum possible number of sets.

**Examples**

**input**

11 11 5

**output**

1

**input**

11 2 3

**output**

-1

**Note**

Note that the rules of the game in this problem differ from the real table tennis game, for example, the rule of "balance" (the winning player has to be at least two points ahead to win a set) has no power within the present problem.

B. Code obfuscation

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Kostya likes Codeforces contests very much. However, he is very disappointed that his solutions are frequently hacked. That's why he decided to obfuscate (intentionally make less readable) his code before upcoming contest.

To obfuscate the code, Kostya first looks at the first variable name used in his program and replaces all its occurrences with a single symbol *a*, then he looks at the second variable name that has not been replaced yet, and replaces all its occurrences with *b*, and so on. Kostya is well-mannered, so he doesn't use any one-letter names before obfuscation. Moreover, there are at most 26 unique identifiers in his programs.

You are given a list of identifiers of some program with removed spaces and line breaks. Check if this program can be a result of Kostya's obfuscation.

**Input**

In the only line of input there is a string *S* of lowercase English letters (1 ≤ |*S*| ≤ 500) — the identifiers of a program with removed whitespace characters.

**Output**

If this program can be a result of Kostya's obfuscation, print "YES" (without quotes), otherwise print "NO".

**Examples**

**input**

abacaba

**output**

YES

**input**

jinotega

**output**

NO

**Note**

In the first sample case, one possible list of identifiers would be "number string number character number string number". Here how Kostya would obfuscate the program:

* replace all occurences of number with a, the result would be "a string a character a string a",
* replace all occurences of string with b, the result would be "a b a character a b a",
* replace all occurences of character with c, the result would be "a b a c a b a",
* all identifiers have been replaced, thus the obfuscation is finished.

A. Neverending competitions

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

There are literally dozens of snooker competitions held each year, and team Jinotega tries to attend them all (for some reason they prefer name "snookah")! When a competition takes place somewhere far from their hometown, Ivan, Artsem and Konstantin take a flight to the contest and back.

Jinotega's best friends, team Base have found a list of their itinerary receipts with information about departure and arrival airports. Now they wonder, where is Jinotega now: at home or at some competition far away? They know that:

* this list contains all Jinotega's flights in this year (**in arbitrary order**),
* Jinotega has only flown from his hometown to a snooker contest and back,
* after each competition Jinotega flies back home (though they may attend a competition in one place several times),
* and finally, at the beginning of the year Jinotega was at home.

Please help them to determine Jinotega's location!

**Input**

In the first line of input there is a single integer *n*: the number of Jinotega's flights (1 ≤ *n* ≤ 100). In the second line there is a string of 3capital Latin letters: the name of Jinotega's home airport. In the next *n* lines there is flight information, one flight per line, in form "XXX->YYY", where "XXX" is the name of departure airport "YYY" is the name of arrival airport. Exactly one of these airports is Jinotega's home airport.

It is guaranteed that flights information is consistent with the knowledge of Jinotega's friends, which is described in the main part of the statement.

**Output**

If Jinotega is now at home, print "home" (without quotes), otherwise print "contest".

**Examples**

**input**

4  
SVO  
SVO->CDG  
LHR->SVO  
SVO->LHR  
CDG->SVO

**output**

home

**input**

3  
SVO  
SVO->HKT  
HKT->SVO  
SVO->RAP

**output**

contest

**Note**

In the first sample Jinotega might first fly from SVO to CDG and back, and then from SVO to LHR and back, so now they should be at home. In the second sample Jinotega must now be at RAP because a flight from RAP back to SVO is not on the list

B. Timofey and cubes

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Young Timofey has a birthday today! He got kit of *n* cubes as a birthday present from his parents. Every cube has a number *ai*, which is written on it. Timofey put all the cubes in a row and went to unpack other presents.

In this time, Timofey's elder brother, Dima reordered the cubes using the following rule. Suppose the cubes are numbered from 1 to *n* in their order. Dima performs several steps, on step *i* he reverses the segment of cubes from *i*-th to (*n* - *i* + 1)-th. He does this while *i* ≤ *n* - *i* + 1.

After performing the operations Dima went away, being very proud of himself. When Timofey returned to his cubes, he understood that their order was changed. Help Timofey as fast as you can and save the holiday — restore the initial order of the cubes using information of their current location.

**Input**

The first line contains single integer *n* (1 ≤ *n* ≤ 2·105) — the number of cubes.

The second line contains *n* integers *a*1, *a*2, ..., *an* ( - 109 ≤ *ai* ≤ 109), where *ai* is the number written on the *i*-th cube after Dima has changed their order.

**Output**

Print *n* integers, separated by spaces — the numbers written on the cubes in their initial order.

It can be shown that the answer is unique.

**Examples**

**input**

7  
4 3 7 6 9 1 2

**output**

2 3 9 6 7 1 4

**input**

8  
6 1 4 2 5 6 9 2

**output**

2 1 6 2 5 4 9 6

**Note**

Consider the first sample.

1. At the begining row was [2, 3, 9, 6, 7, 1, 4].
2. After first operation row was [4, 1, 7, 6, 9, 3, 2].
3. After second operation row was [4, 3, 9, 6, 7, 1, 2].
4. After third operation row was [4, 3, 7, 6, 9, 1, 2].
5. At fourth operation we reverse just middle element, so nothing has changed. The final row is [4, 3, 7, 6, 9, 1, 2]. So the answer for this case is row [2, 3, 9, 6, 7, 1, 4].

A. Taymyr is calling you

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Comrade Dujikov is busy choosing artists for Timofey's birthday and is recieving calls from Taymyr from Ilia-alpinist.

Ilia-alpinist calls every *n* minutes, i.e. in minutes *n*, 2*n*, 3*n* and so on. Artists come to the comrade every *m* minutes, i.e. in minutes *m*, 2*m*, 3*m* and so on. The day is *z* minutes long, i.e. the day consists of minutes 1, 2, ..., *z*. How many artists should be killed so that there are no artists in the room when Ilia calls? Consider that a call and a talk with an artist take exactly one minute.

**Input**

The only string contains three integers — *n*, *m* and *z* (1 ≤ *n*, *m*, *z* ≤ 104).

**Output**

Print single integer — the minimum number of artists that should be killed so that there are no artists in the room when Ilia calls.

**Examples**

**input**

1 1 10

**output**

10

**input**

1 2 5

**output**

2

**input**

2 3 9

**output**

1

**Note**

Taymyr is a place in the north of Russia.

In the first test the artists come each minute, as well as the calls, so we need to kill all of them.

In the second test we need to kill artists which come on the second and the fourth minutes.

In the third test — only the artist which comes on the sixth minute.

E. Timofey and our friends animals

time limit per test

7 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

After his birthday party, Timofey went to his favorite tree alley in a park. He wants to feed there his favorite birds — crows.

It's widely known that each tree is occupied by a single crow family. The trees in the alley form a row and are numbered from 1 to *n*. Some families are friends to each other. For some reasons, two families can be friends only if they live not too far from each other, more precisely, there is no more than *k* - 1 trees between any pair of friend families. Formally, the family on the *u*-th tree and the family on the *v*-th tree can be friends only if |*u* - *v*| ≤ *k* holds.

One of the friendship features is that if some family learns that Timofey is feeding crows somewhere, it notifies about this all friend families. Thus, after Timofey starts to feed crows under some tree, all the families that are friends to the family living on this tree, as well as their friends and so on, fly to the feeding place. Of course, the family living on the tree also comes to the feeding place.

Today Timofey came to the alley and noticed that all the families that live on trees with numbers strictly less than *l* or strictly greater than *r* have flown away. Thus, it is not possible to pass the information about feeding through them. Moreover, there is no need to feed them. Help Timofey to learn what is the minimum number of trees under which he has to feed crows so that all the families that have remained will get the information about feeding. You are given several situations, described by integers *l* and *r*, you need to calculate the answer for all of them.

**Input**

The first line contains integers *n* and *k* (1 ≤ *n* ≤ 105, 1 ≤ *k* ≤ 5), where *n* is the number of trees, and *k* is the maximum possible distance between friend families.

The next line contains single integer *m* (0 ≤ *m* ≤ *n*·*k*) — the number of pair of friend families.

Each of the next *m* lines contains two integers *u* and *v* (1 ≤ *u*, *v* ≤ 105), that means that the families on trees *u* and *v* are friends. It is guaranteed that *u* ≠ *v* and |*u* - *v*| ≤ *k*. All the given pairs are distinct.

The next line contains single integer *q* (1 ≤ *q* ≤ 105) — the number of situations you need to calculate the answer in.

Each of the next *q* lines contains two integers *l* and *r* (1 ≤ *l* ≤ *r* ≤ 105), that means that in this situation families that have flown away lived on such trees *x*, so that either *x* < *l* or *x* > *r*.

**Output**

Print *q* lines. Line *i* should contain single integer — the answer in the *i*-th situation.

**Example**

**input**

5 3  
3  
1 3  
2 3  
4 5  
5  
1 1  
1 2  
2 3  
1 3  
1 5

**output**

1  
2  
1  
1  
2

**Note**

In the first example the following family pairs are friends: (1, 3), (2, 3) and (4, 5).

* In the first situation only the first family has remained, so the answer is 1.
* In the second situation the first two families have remained, and they aren't friends, so the answer is 2.
* In the third situation the families 2 and 3 are friends, so it is enough to feed any of them, the answer is 1.
* In the fourth situation we can feed the first family, then the third family will get the information from the first family, and the second family will get the information from the third. The answer is 1.
* In the fifth situation we can feed the first and the fifth families, so the answer is 2.

D. Timofey and a flat tree

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Little Timofey has a big tree — an undirected connected graph with *n* vertices and no simple cycles. He likes to walk along it. His tree is flat so when he walks along it he sees it entirely. Quite naturally, when he stands on a vertex, he sees the tree as a rooted tree with the root in this vertex.

Timofey assumes that the **more** non-isomorphic subtrees are there in the tree, the more beautiful the tree is. A subtree of a vertex is a subgraph containing this vertex and all its descendants. You should tell Timofey the vertex in which he should stand to see the most beautiful rooted tree.

Subtrees of vertices *u* and *v* are isomorphic if the number of children of *u* equals the number of children of *v*, and their children can be arranged in such a way that the subtree of the first son of *u* is isomorphic to the subtree of the first son of *v*, the subtree of the second son of *u* is isomorphic to the subtree of the second son of *v*, and so on. In particular, subtrees consisting of single vertex are isomorphic to each other.

**Input**

First line contains single integer *n* (1 ≤ *n* ≤ 105) — number of vertices in the tree.

Each of the next *n* - 1 lines contains two integers *ui* and *vi* (1 ≤ *ui*, *vi* ≤ 105, *ui* ≠ *vi*), denoting the vertices the *i*-th edge connects.

It is guaranteed that the given graph is a tree.

**Output**

Print single integer — the index of the vertex in which Timofey should stand. If there are many answers, you can print any of them.

**Examples**

**input**

3  
1 2  
2 3

**output**

1

**input**

7  
1 2  
4 2  
2 3  
5 6  
6 7  
3 7

**output**

1

**input**

10  
1 7  
1 8  
9 4  
5 1  
9 2  
3 5  
10 6  
10 9  
5 10

**output**

2

**Note**

In the first example we can stand in the vertex 1 or in the vertex 3 so that every subtree is non-isomorphic. If we stand in the vertex 2, then subtrees of vertices 1 and 3 are isomorphic.

In the second example, if we stand in the vertex 1, then only subtrees of vertices 4 and 5 are isomorphic.

In the third example, if we stand in the vertex 1, then subtrees of vertices 2, 3, 4, 6, 7 and 8 are isomorphic. If we stand in the vertex 2, than only subtrees of vertices 3, 4, 6, 7 and 8 are isomorphic. If we stand in the vertex 5, then subtrees of vertices 2, 3, 4, 6, 7 and 8are isomorphic, and subtrees of vertices 1 and 9 are isomorphic as well:

1 9  
 /\ /\  
7 8 4 2

C. Timofey and remoduling

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Little Timofey likes integers a lot. Unfortunately, he is very young and can't work with very big integers, so he does all the operations modulo his favorite prime *m*. Also, Timofey likes to look for arithmetical progressions everywhere.

One of his birthday presents was a sequence of **distinct** integers *a*1, *a*2, ..., *an*. Timofey wants to know whether he can rearrange the elements of the sequence so that is will be an arithmetical progression modulo *m*, or not.

Arithmetical progression modulo *m* of length *n* with first element *x* and difference *d* is sequence of integers *x*, *x* + *d*, *x* + 2*d*, ..., *x* + (*n* - 1)·*d*, each taken modulo *m*.

**Input**

The first line contains two integers *m* and *n* (2 ≤ *m* ≤ 109 + 7, 1 ≤ *n* ≤ 105, *m* is prime) — Timofey's favorite prime module and the length of the sequence.

The second line contains *n* **distinct** integers *a*1, *a*2, ..., *an* (0 ≤ *ai* < *m*) — the elements of the sequence.

**Output**

Print -1 if it is not possible to rearrange the elements of the sequence so that is will be an arithmetical progression modulo *m*.

Otherwise, print two integers — the first element of the obtained progression *x* (0 ≤ *x* < *m*) and its difference *d* (0 ≤ *d* < *m*).

If there are multiple answers, print any of them.

**Examples**

**input**

17 5  
0 2 4 13 15

**output**

13 2

**input**

17 5  
0 2 4 13 14

**output**

-1

**input**

5 3  
1 2 3

**output**

3 4

B. Timofey and rectangles

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

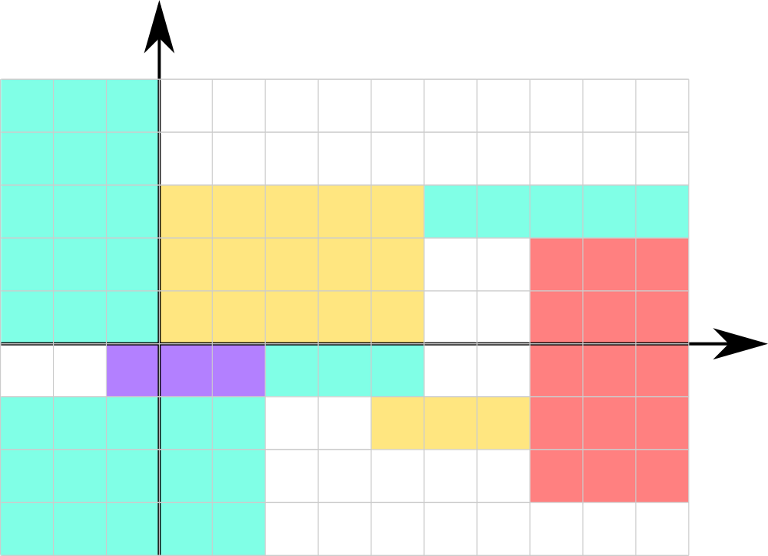
output

standard output

One of Timofey's birthday presents is a colourbook in a shape of an infinite plane. On the plane *n* rectangles with sides parallel to coordinate axes are situated. All sides of the rectangles have **odd** length. Rectangles cannot intersect, but they can touch each other.

Help Timofey to color his rectangles in 4 different colors in such a way that every two rectangles touching each other by side would have different color, or determine that it is impossible.

Two rectangles intersect if their intersection has positive area. Two rectangles touch by sides if there is a pair of sides such that their intersection has non-zero length

The picture corresponds to the first example

**Input**

The first line contains single integer *n* (1 ≤ *n* ≤ 5·105) — the number of rectangles.

*n* lines follow. The *i*-th of these lines contains four integers *x*1, *y*1, *x*2 and *y*2 ( - 109 ≤ *x*1 < *x*2 ≤ 109,  - 109 ≤ *y*1 < *y*2 ≤ 109), that means that points (*x*1, *y*1) and (*x*2, *y*2) are the coordinates of two opposite corners of the *i*-th rectangle.

It is guaranteed, that all sides of the rectangles have **odd** lengths and rectangles don't intersect each other.

**Output**

Print "NO" in the only line if it is impossible to color the rectangles in 4 different colors in such a way that every two rectangles touching each other by side would have different color.

Otherwise, print "YES" in the first line. Then print *n* lines, in the *i*-th of them print single integer *ci* (1 ≤ *ci* ≤ 4) — the color of *i*-th rectangle.

**Example**

**input**

8  
0 0 5 3  
2 -1 5 0  
-3 -4 2 -1  
-1 -1 2 0  
-3 0 0 5  
5 2 10 3  
7 -3 10 2  
4 -2 7 -1

**output**

YES  
1  
2  
2  
3  
2  
2  
4  
1

A. Timofey and a tree

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Each New Year Timofey and his friends cut down a tree of *n* vertices and bring it home. After that they paint all the *n* its vertices, so that the *i*-th vertex gets color *ci*.

Now it's time for Timofey birthday, and his mother asked him to remove the tree. Timofey removes the tree in the following way: he takes some vertex in hands, while all the other vertices move down so that the tree becomes rooted at the chosen vertex. After that Timofey brings the tree to a trash can.

Timofey doesn't like it when many colors are mixing together. A subtree annoys him if there are vertices of different color in it. Timofey wants to find a vertex which he should take in hands so that there are no subtrees that annoy him. He doesn't consider the whole tree as a subtree since he can't see the color of the root vertex.

A subtree of some vertex is a subgraph containing that vertex and all its descendants.

Your task is to determine if there is a vertex, taking which in hands Timofey wouldn't be annoyed.

**Input**

The first line contains single integer *n* (2 ≤ *n* ≤ 105) — the number of vertices in the tree.

Each of the next *n* - 1 lines contains two integers *u* and *v* (1 ≤ *u*, *v* ≤ *n*, *u* ≠ *v*), denoting there is an edge between vertices *u* and *v*. It is guaranteed that the given graph is a tree.

The next line contains *n* integers *c*1, *c*2, ..., *cn* (1 ≤ *ci* ≤ 105), denoting the colors of the vertices.

**Output**

Print "NO" in a single line, if Timofey can't take the tree in such a way that it doesn't annoy him.

Otherwise print "YES" in the first line. In the second line print the index of the vertex which Timofey should take in hands. If there are multiple answers, print any of them.

**Examples**

**input**

4  
1 2  
2 3  
3 4  
1 2 1 1

**output**

YES  
2

**input**

3  
1 2  
2 3  
1 2 3

**output**

YES  
2

**input**

4  
1 2  
2 3  
3 4  
1 2 1 2

**output**

NO

F. Tree nesting

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given two trees (connected undirected acyclic graphs) *S* and *T*.

Count the number of subtrees (connected subgraphs) of *S* that are isomorphic to tree *T*. Since this number can get quite large, output it modulo 109 + 7.

Two subtrees of tree *S* are considered different, if there exists a vertex in *S* that belongs to exactly one of them.

Tree *G* is called isomorphic to tree *H* if there exists a bijection *f* from the set of vertices of *G* to the set of vertices of *H* that has the following property: if there is an edge between vertices *A* and *B* in tree *G*, then there must be an edge between vertices *f*(*A*) and *f*(*B*) in tree *H*. And vice versa — if there is an edge between vertices *A* and *B* in tree *H*, there must be an edge between *f*- 1(*A*) and *f*- 1(*B*) in tree *G*.

**Input**

The first line contains a single integer |*S*| (1 ≤ |*S*| ≤ 1000) — the number of vertices of tree *S*.

Next |*S*| - 1 lines contain two integers *ui* and *vi* (1 ≤ *ui*, *vi* ≤ |*S*|) and describe edges of tree *S*.

The next line contains a single integer |*T*| (1 ≤ |*T*| ≤ 12) — the number of vertices of tree *T*.

Next |*T*| - 1 lines contain two integers *xi* and *yi* (1 ≤ *xi*, *yi* ≤ |*T*|) and describe edges of tree *T*.

**Output**

On the first line output a single integer — the answer to the given task modulo 109 + 7.

**Examples**

**input**

5  
1 2  
2 3  
3 4  
4 5  
3  
1 2  
2 3

**output**

3

**input**

3  
2 3  
3 1  
3  
1 2  
1 3

**output**

1

**input**

7  
1 2  
1 3  
1 4  
1 5  
1 6  
1 7  
4  
4 1  
4 2  
4 3

**output**

20

**input**

5  
1 2  
2 3  
3 4  
4 5  
4  
4 1  
4 2  
4 3

**output**

0

E. Radio stations

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

In the lattice points of the coordinate line there are *n* radio stations, the *i*-th of which is described by three integers:

* *xi* — the coordinate of the *i*-th station on the line,
* *ri* — the broadcasting range of the *i*-th station,
* *fi* — the broadcasting frequency of the *i*-th station.

We will say that two radio stations with numbers *i* and *j* reach each other, if the broadcasting range of each of them is more or equal to the distance between them. In other words *min*(*ri*, *rj*) ≥ |*xi* - *xj*|.

Let's call a pair of radio stations (*i*, *j*) bad if *i* < *j*, stations *i* and *j* reach each other and they are close in frequency, that is, |*fi* - *fj*| ≤ *k*.

Find the number of bad pairs of radio stations.

**Input**

The first line contains two integers *n* and *k* (1 ≤ *n* ≤ 105, 0 ≤ *k* ≤ 10) — the number of radio stations and the maximum difference in the frequencies for the pair of stations that reach each other to be considered bad.

In the next *n* lines follow the descriptions of radio stations. Each line contains three integers *xi*, *ri* and *fi* (1 ≤ *xi*, *ri* ≤ 109, 1 ≤ *fi* ≤ 104) — the coordinate of the *i*-th radio station, it's broadcasting range and it's broadcasting frequency. **No two radio stations will share a coordinate**.

**Output**

Output the number of bad pairs of radio stations.

**Examples**

**input**

3 2  
1 3 10  
3 2 5  
4 10 8

**output**

1

**input**

3 3  
1 3 10  
3 2 5  
4 10 8

**output**

2

**input**

5 1  
1 3 2  
2 2 4  
3 2 1  
4 2 1  
5 3 3

**output**

2

**input**

5 1  
1 5 2  
2 5 4  
3 5 1  
4 5 1  
5 5 3

**output**

5

D. Maximum path

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a rectangular table 3 × *n*. Each cell contains an integer. You can move from one cell to another if they share a side.

Find such path from the upper left cell to the bottom right cell of the table that doesn't visit any of the cells twice, and the sum of numbers written in the cells of this path is maximum possible.

**Input**

The first line contains an integer *n* (1 ≤ *n* ≤ 105)  — the number of columns in the table.

Next three lines contain *n* integers each  — the description of the table. The *j*-th number in the *i*-th line corresponds to the cell *aij*( - 109 ≤ *aij* ≤ 109) of the table.

**Output**

Output the maximum sum of numbers on a path from the upper left cell to the bottom right cell of the table, that doesn't visit any of the cells twice.

**Examples**

**input**

3  
1 1 1  
1 -1 1  
1 1 1

**output**

7

**input**

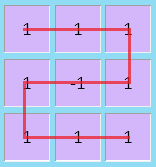
5  
10 10 10 -1 -1  
-1 10 10 10 10  
-1 10 10 10 10

**output**

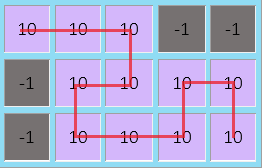
110

**Note**

The path for the first example:



The path for the second example:



C. Two strings

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given two strings *a* and *b*. You have to remove the minimum possible number of **consecutive** (standing one after another) characters from string *b* in such a way that it becomes a subsequence of string *a*. It can happen that you will not need to remove any characters at all, or maybe you will have to remove all of the characters from *b* and make it empty.

Subsequence of string *s* is any such string that can be obtained by erasing zero or more characters (**not necessarily consecutive**) from string *s*.

**Input**

The first line contains string *a*, and the second line — string *b*. Both of these strings are nonempty and consist of lowercase letters of English alphabet. The length of each string is no bigger than 105 characters.

**Output**

On the first line output a subsequence of string *a*, obtained from *b* by erasing the minimum number of consecutive characters.

If the answer consists of zero characters, output «-» (a minus sign).

**Examples**

**input**

hi  
bob

**output**

-

**input**

abca  
accepted

**output**

ac

**input**

abacaba  
abcdcba

**output**

abcba

**Note**

In the first example strings *a* and *b* don't share any symbols, so the longest string that you can get is empty.

In the second example ac is a subsequence of *a*, and at the same time you can obtain it by erasing consecutive symbols cepted from string *b*.

B. USB vs. PS/2

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Due to the increase in the number of students of Berland State University it was decided to equip a new computer room. You were given the task of buying mouses, and you have to spend as little as possible. After all, the country is in crisis!

The computers bought for the room were different. Some of them had only USB ports, some — only PS/2 ports, and some had both options.

You have found a price list of a certain computer shop. In it, for *m* mouses it is specified the cost and the type of the port that is required to plug the mouse in (USB or PS/2). Each mouse from the list can be bought at most once.

You want to buy some set of mouses from the given price list in such a way so that you maximize the number of computers equipped with mouses (it is not guaranteed that you will be able to equip all of the computers), and in case of equality of this value you want to minimize the total cost of mouses you will buy.

**Input**

The first line contains three integers *a*, *b* and *c* (0 ≤ *a*, *b*, *c* ≤ 105)  — the number of computers that only have USB ports, the number of computers, that only have PS/2 ports, and the number of computers, that have both options, respectively.

The next line contains one integer *m* (0 ≤ *m* ≤ 3·105)  — the number of mouses in the price list.

The next *m* lines each describe another mouse. The *i*-th line contains first integer *vali* (1 ≤ *vali* ≤ 109)  — the cost of the *i*-th mouse, then the type of port (USB or PS/2) that is required to plug the mouse in.

**Output**

Output two integers separated by space — the number of equipped computers and the total cost of the mouses you will buy.

**Example**

**input**

2 1 1  
4  
5 USB  
6 PS/2  
3 PS/2  
7 PS/2

**output**

3 14

**Note**

In the first example you can buy the first three mouses. This way you will equip one of the computers that has only a USB port with a USB mouse, and the two PS/2 mouses you will plug into the computer with PS/2 port and the computer with both ports.

A. k-th divisor

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given two integers *n* and *k*. Find *k*-th smallest divisor of *n*, or report that it doesn't exist.

Divisor of *n* is any such natural number, that *n* can be divided by it without remainder.

**Input**

The first line contains two integers *n* and *k* (1 ≤ *n* ≤ 1015, 1 ≤ *k* ≤ 109).

**Output**

If *n* has less than *k* divisors, output -1.

Otherwise, output the *k*-th smallest divisor of *n*.

**Examples**

**input**

4 2

**output**

2

**input**

5 3

**output**

-1

**input**

12 5

**output**

6

**Note**

In the first example, number 4 has three divisors: 1, 2 and 4. The second one is 2.

In the second example, number 5 has only two divisors: 1 and 5. The third divisor doesn't exist, so the answer is -1.

F. Dasha and Photos

time limit per test

4 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Dasha decided to have a rest after solving the problem *D* and began to look photos from previous competitions.

Let's call photos as the matrix with the size *n* × *m*, which consists of lowercase English letters.

Some *k* photos especially interested her, because they can be received from photo-template by painting a rectangular area in a certain color. Let's call such photos special.



More formally the *i*-th special photo is received from the photo-template by replacing all characters on some rectangle with upper left corner of the cell with coordinates (*ai*, *bi*) and lower right corner in the cell with coordinates (*ci*, *di*) to the symbol *ei*.

Dasha asks you to find the special photo so that the total distance from it to all other special photos is minimum. And calculate this distance.

Determine the distance between two photos as the sum of distances between all corresponding letters. The distance between two letters is the difference module of their positions in the alphabet. For example, the distance between letters 'h' and 'm' equals |8 - 13| = 5, because the letter 'h' is the 8-th in the alphabet, the letter 'm' is the 13-th.

**Input**

The first line contains three integers *n*, *m*, *k* (1 ≤ *n*, *m* ≤ 103, 1 ≤ *k* ≤ 3·105) — the number of strings in the photo-template, the number of columns and the number of special photos which are interesting for Dasha.

The next *n* lines contains the string with *m* length which consists of little Latin characters — the description of the photo-template.

Each of the next *k* lines contains the description of the special photo in the following format, "*ai* *bi* *ci* *di* *ei*" (1 ≤ *ai* ≤ *ci* ≤ *n*, 1 ≤ *bi* ≤ *di* ≤ *m*), where (*ai*, *bi*) — is the coordinate of the upper left corner of the rectangle, (*ci*, *di*) — is the description of the lower right corner, and *ei* — is the little Latin letter which replaces the photo-template in the described rectangle.

**Output**

In the only line print the minimum total distance from the found special photo to all other special photos.

**Examples**

**input**

3 3 2  
aaa  
aaa  
aaa  
1 1 2 2 b  
2 2 3 3 c

**output**

10

**input**

5 5 3  
abcde  
eabcd  
deabc  
cdeab  
bcdea  
1 1 3 4 f  
1 2 3 3 e  
1 3 3 4 i

**output**

59

**Note**

In the first example the photos are following:

bba aaa  
bba acc  
aaa acc

The distance between them is 10.

E. Dasha and Puzzle

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Dasha decided to have a rest after solving the problem. She had been ready to start her favourite activity — origami, but remembered the puzzle that she could not solve.

The tree is a non-oriented connected graph without cycles. In particular, there always are *n* - 1 edges in a tree with *n* vertices.

The puzzle is to position the vertices at the points of the Cartesian plane with integral coordinates, so that the segments between the vertices connected by edges are parallel to the coordinate axes. Also, the intersection of segments is allowed only at their ends. Distinct vertices should be placed at different points.

Help Dasha to find any suitable way to position the tree vertices on the plane.

It is guaranteed that if it is possible to position the tree vertices on the plane without violating the condition which is given above, then you can do it by using points with integral coordinates which don't exceed 1018 in absolute value.

**Input**

The first line contains single integer *n* (1 ≤ *n* ≤ 30) — the number of vertices in the tree.

Each of next *n* - 1 lines contains two integers *ui*, *vi* (1 ≤ *ui*, *vi* ≤ *n*) that mean that the *i*-th edge of the tree connects vertices *ui* and *vi*.

It is guaranteed that the described graph is a tree.

**Output**

If the puzzle doesn't have a solution then in the only line print "NO".

Otherwise, the first line should contain "YES". The next *n* lines should contain the pair of integers *xi*, *yi* (|*xi*|, |*yi*| ≤ 1018) — the coordinates of the point which corresponds to the *i*-th vertex of the tree.

If there are several solutions, print any of them.

**Examples**

**input**

7  
1 2  
1 3  
2 4  
2 5  
3 6  
3 7

**output**

YES  
0 0  
1 0  
0 1  
2 0  
1 -1  
-1 1  
0 2

**input**

6  
1 2  
2 3  
2 4  
2 5  
2 6

**output**

NO

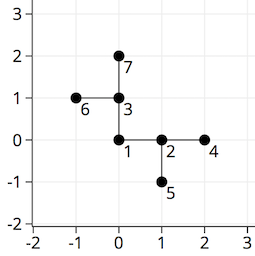
**input**

4  
1 2  
2 3  
3 4

**output**

YES  
3 3  
4 3  
5 3  
6 3

**Note**

In the first sample one of the possible positions of tree is:

D. Dasha and Very Difficult Problem

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Dasha logged into the system and began to solve problems. One of them is as follows:

Given two sequences *a* and *b* of length *n* each you need to write a sequence *c* of length *n*, the *i*-th element of which is calculated as follows: *ci* = *bi* - *ai*.

About sequences *a* and *b* we know that their elements are in the range from *l* to *r*. More formally, elements satisfy the following conditions: *l* ≤ *ai* ≤ *r* and *l* ≤ *bi* ≤ *r*. About sequence *c* we know that all its elements are distinct.

Dasha wrote a solution to that problem quickly, but checking her work on the standard test was not so easy. Due to an error in the test system only the sequence *a* and the *compressed sequence* of the sequence *c* were known from that test.

Let's give the definition to a *compressed sequence*. A *compressed sequence* of sequence *c* of length *n* is a sequence *p* of length *n*, so that *pi* equals to the number of integers which are less than or equal to *ci* in the sequence *c*. For example, for the sequence *c* = [250, 200, 300, 100, 50] the compressed sequence will be *p* = [4, 3, 5, 2, 1]. Pay attention that in *c* all integers are distinct. Consequently, the *compressed sequence* contains all integers from 1 to *n* inclusively.

Help Dasha to find any sequence *b* for which the calculated *compressed sequence* of sequence *c* is correct.

**Input**

The first line contains three integers *n*, *l*, *r* (1 ≤ *n* ≤ 105, 1 ≤ *l* ≤ *r* ≤ 109) — the length of the sequence and boundaries of the segment where the elements of sequences *a* and *b* are.

The next line contains *n* integers *a*1,  *a*2,  ...,  *an* (*l* ≤ *ai* ≤ *r*) — the elements of the sequence *a*.

The next line contains *n* distinct integers *p*1,  *p*2,  ...,  *pn* (1 ≤ *pi* ≤ *n*) — the *compressed sequence* of the sequence *c*.

**Output**

If there is no the suitable sequence *b*, then in the only line print "-1".

Otherwise, in the only line print *n* integers — the elements of any suitable sequence *b*.

**Examples**

**input**

5 1 5  
1 1 1 1 1  
3 1 5 4 2

**output**

3 1 5 4 2

**input**

4 2 9  
3 4 8 9  
3 2 1 4

**output**

2 2 2 9

**input**

6 1 5  
1 1 1 1 1 1  
2 3 5 4 1 6

**output**

-1

**Note**

Sequence *b* which was found in the second sample is suitable, because calculated sequence *c* = [2 - 3, 2 - 4, 2 - 8, 9 - 9] = [ - 1,  - 2,  - 6, 0] (note that *ci* = *bi* - *ai*) has compressed sequence equals to *p* = [3, 2, 1, 4].

C. Dasha and Password

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

After overcoming the stairs Dasha came to classes. She needed to write a password to begin her classes. The password is a string of length *n* which satisfies the following requirements:

* There is at least one digit in the string,
* There is at least one lowercase (small) letter of the Latin alphabet in the string,
* There is at least one of three listed symbols in the string: '#', '\*', '&'.



Considering that these are programming classes it is not easy to write the password.

For each character of the password we have a fixed string of length *m*, on each of these *n* strings there is a pointer on some character. The *i*-th character displayed on the screen is the pointed character in the *i*-th string. Initially, all pointers are on characters with indexes 1 in the corresponding strings (all positions are numbered starting from one).

During one operation Dasha can move a pointer in one string one character to the left or to the right. Strings are cyclic, it means that when we move the pointer which is on the character with index 1 to the left, it moves to the character with the index *m*, and when we move it to the right from the position *m* it moves to the position 1.

You need to determine the minimum number of operations necessary to make the string displayed on the screen a valid password.

**Input**

The first line contains two integers *n*, *m* (3 ≤ *n* ≤ 50, 1 ≤ *m* ≤ 50) — the length of the password and the length of strings which are assigned to password symbols.

Each of the next *n* lines contains the string which is assigned to the *i*-th symbol of the password string. Its length is *m*, it consists of digits, lowercase English letters, and characters '#', '\*' or '&'.

You have such input data that you can always get a valid password.

**Output**

Print one integer — the minimum number of operations which is necessary to make the string, which is displayed on the screen, a valid password.

**Examples**

**input**

3 4  
1\*\*2  
a3\*0  
c4\*\*

**output**

1

**input**

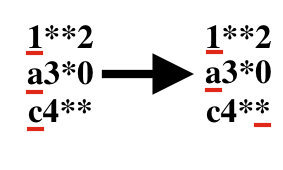
5 5  
#\*&#\*  
\*a1c&  
&q2w\*  
#a3c#  
\*&#\*&

**output**

3

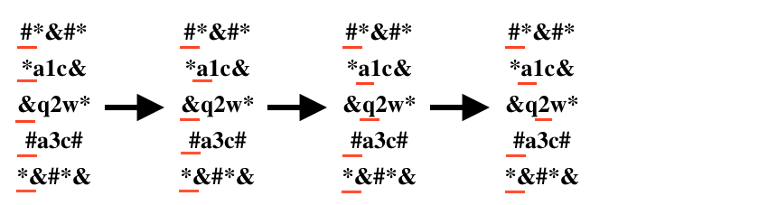
**Note**

In the first test it is necessary to move the pointer of the third string to one left to get the optimal answer.



In the second test one of possible algorithms will be:

* to move the pointer of the second symbol once to the right.
* to move the pointer of the third symbol twice to the right.



B. Dasha and friends

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

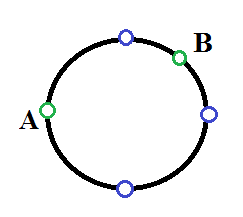
output

standard output

Running with barriers on the circle track is very popular in the country where Dasha lives, so no wonder that on her way to classes she saw the following situation:

The track is the circle with length *L*, in distinct points of which there are *n* barriers. Athlete always run the track in counterclockwise direction if you look on him from above. All barriers are located at integer distance from each other along the track.

Her friends the parrot Kefa and the leopard Sasha participated in competitions and each of them ran one lap. Each of the friends started from some integral point on the track. Both friends wrote the distance from their start along the track to each of the *n* barriers. Thus, each of them wrote *n* integers in the ascending order, each of them was between 0 and *L* - 1, inclusively.

Consider an example. Let *L* = 8, blue points are barriers, and green points are Kefa's start (A) and Sasha's start (B). Then Kefa writes down the sequence[2, 4, 6], and Sasha writes down [1, 5, 7].

There are several tracks in the country, all of them have same length and same number of barriers, but the positions of the barriers can differ among different tracks. Now Dasha is interested if it is possible that Kefa and Sasha ran the same track or they participated on different tracks.

Write the program which will check that Kefa's and Sasha's tracks coincide (it means that one can be obtained from the other by changing the start position). Note that they always run the track in one direction — counterclockwise, if you look on a track from above.

**Input**

The first line contains two integers *n* and *L* (1 ≤ *n* ≤ 50, *n* ≤ *L* ≤ 100) — the number of barriers on a track and its length.

The second line contains *n* distinct integers in the ascending order — the distance from Kefa's start to each barrier in the order of its appearance. All integers are in the range from 0 to *L* - 1 inclusively.

The second line contains *n* distinct integers in the ascending order — the distance from Sasha's start to each barrier in the order of its overcoming. All integers are in the range from 0 to *L* - 1 inclusively.

**Output**

Print "YES" (without quotes), if Kefa and Sasha ran the coinciding tracks (it means that the position of all barriers coincides, if they start running from the same points on the track). Otherwise print "NO" (without quotes).

**Examples**

**input**

3 8  
2 4 6  
1 5 7

**output**

YES

**input**

4 9  
2 3 5 8  
0 1 3 6

**output**

YES

**input**

2 4  
1 3  
1 2

**output**

NO

**Note**

The first test is analyzed in the statement.

A. Dasha and Stairs

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

On her way to programming school tiger Dasha faced her first test — a huge staircase!



The steps were numbered from one to infinity. As we know, tigers are very fond of all striped things, it is possible that it has something to do with their color. So on some interval of her way she calculated two values — the number of steps with even and odd numbers.

You need to check whether there is an interval of steps from the *l*-th to the *r*-th (1 ≤ *l* ≤ *r*), for which values that Dasha has found are correct.

**Input**

In the only line you are given two integers *a*, *b* (0 ≤ *a*, *b* ≤ 100) — the number of even and odd steps, accordingly.

**Output**

In the only line print "YES", if the interval of steps described above exists, and "NO" otherwise.

**Examples**

**input**

2 3

**output**

YES

**input**

3 1

**output**

NO

**Note**

In the first example one of suitable intervals is from 1 to 5. The interval contains two even steps — 2 and 4, and three odd: 1, 3 and 5.

B. Frodo and pillows

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

*n* hobbits are planning to spend the night at Frodo's house. Frodo has *n* beds standing in a row and *m* pillows (*n* ≤ *m*). Each hobbit needs a bed and at least one pillow to sleep, however, everyone wants as many pillows as possible. Of course, it's not always possible to share pillows equally, but any hobbit gets hurt if he has at least two pillows less than some of his neighbors have.

Frodo will sleep on the *k*-th bed in the row. What is the maximum number of pillows he can have so that every hobbit has at least one pillow, every pillow is given to some hobbit and no one is hurt?

**Input**

The only line contain three integers *n*, *m* and *k* (1 ≤ *n* ≤ *m* ≤ 109, 1 ≤ *k* ≤ *n*) — the number of hobbits, the number of pillows and the number of Frodo's bed.

**Output**

Print single integer — the maximum number of pillows Frodo can have so that no one is hurt.

**Examples**

**input**

4 6 2

**output**

2

**input**

3 10 3

**output**

4

**input**

3 6 1

**output**

3

**Note**

In the first example Frodo can have at most two pillows. In this case, he can give two pillows to the hobbit on the first bed, and one pillow to each of the hobbits on the third and the fourth beds.

In the second example Frodo can take at most four pillows, giving three pillows to each of the others.

In the third example Frodo can take three pillows, giving two pillows to the hobbit in the middle and one pillow to the hobbit on the third bed.

A. Petr and a calendar

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Petr wants to make a calendar for current month. For this purpose he draws a table in which columns correspond to weeks (a week is seven consequent days from Monday to Sunday), rows correspond to weekdays, and cells contain dates. For example, a calendar for January 2017 should look like on the picture:



Petr wants to know how many columns his table should have given the month and the weekday of the first date of that month? Assume that the year is non-leap.

**Input**

The only line contain two integers *m* and *d* (1 ≤ *m* ≤ 12, 1 ≤ *d* ≤ 7) — the number of month (January is the first month, December is the twelfth) and the weekday of the first date of this month (1 is Monday, 7 is Sunday).

**Output**

Print single integer: the number of columns the table should have.

**Examples**

**input**

1 7

**output**

6

**input**

1 1

**output**

5

**input**

11 6

**output**

5

**Note**

The first example corresponds to the January 2017 shown on the picture in the statements.

In the second example 1-st January is Monday, so the whole month fits into 5 columns.

In the third example 1-st November is Saturday and 5 columns is enough.

F. Geometrical Progression

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

For given *n*, *l* and *r* find the number of distinct geometrical progression, each of which contains *n* distinct integers not less than *l* and not greater than *r*. In other words, for each progression the following must hold: *l* ≤ *ai* ≤ *r* and *ai* ≠ *aj* , where *a*1, *a*2, ..., *an* is the geometrical progression, 1 ≤ *i*, *j* ≤ *n* and *i* ≠ *j*.

Geometrical progression is a sequence of numbers *a*1, *a*2, ..., *an* where each term after first is found by multiplying the previous one by a fixed non-zero number *d* called the common ratio. Note that in our task *d* may be non-integer. For example in progression 4, 6, 9, common ratio is http://codeforces.com/predownloaded/d3/dd/d3ddac9f40d8fa3694ce7f5aa1e64bea5bcfd7f8.png.

Two progressions *a*1, *a*2, ..., *an* and *b*1, *b*2, ..., *bn* are considered different, if there is such *i* (1 ≤ *i* ≤ *n*) that *ai* ≠ *bi*.

**Input**

The first and the only line cotains three integers *n*, *l* and *r* (1 ≤ *n* ≤ 107, 1 ≤ *l* ≤ *r* ≤ 107).

**Output**

Print the integer *K* — is the answer to the problem.

**Examples**

**input**

1 1 10

**output**

10

**input**

2 6 9

**output**

12

**input**

3 1 10

**output**

8

**input**

3 3 10

**output**

2

**Note**

These are possible progressions for the first test of examples:

* 1;
* 2;
* 3;
* 4;
* 5;
* 6;
* 7;
* 8;
* 9;
* 10.

These are possible progressions for the second test of examples:

* 6, 7;
* 6, 8;
* 6, 9;
* 7, 6;
* 7, 8;
* 7, 9;
* 8, 6;
* 8, 7;
* 8, 9;
* 9, 6;
* 9, 7;
* 9, 8.

These are possible progressions for the third test of examples:

* 1, 2, 4;
* 1, 3, 9;
* 2, 4, 8;
* 4, 2, 1;
* 4, 6, 9;
* 8, 4, 2;
* 9, 3, 1;
* 9, 6, 4.

These are possible progressions for the fourth test of examples:

* 4, 6, 9;
* 9, 6, 4.

E. Broken Tree

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a tree that has *n* vertices, which are numbered from 1 to *n*, where the vertex number one is the root. Each edge has weight *wi* and strength *pi*.

Botanist Innokentiy, who is the only member of the jury of the Olympiad in Informatics, doesn't like broken trees.

The tree is broken if there is such an edge the strength of which is less than the sum of weight of subtree's edges to which it leads.

It is allowed to reduce weight of any edge by arbitrary integer value, but then the strength of its edge is reduced by the same value. It means if the weight of the edge is 10, and the strength is 12, then by the reducing the weight by 7 its weight will equal 3, and the strength will equal 5.

It is not allowed to increase the weight of the edge.

Your task is to get the tree, which is not broken, by reducing the weight of edges of the given tree, and also all edged should have the positive weight, moreover, the total weight of all edges should be as large as possible.

It is obvious that the strength of edges can not be negative, however it can equal zero if the weight of the subtree equals zero.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 2·105) — the number of vertices in the tree. The next *n* - 1 lines contains the description of edges. Each line contains four integers *x*, *y*, *w*, *p* (1 ≤ *x*, *y* ≤ *n*, 1 ≤ *w* ≤ 109, 0 ≤ *p* ≤ 109), where *x* and *y* — vertices which connect the edge (the vertex number *x* is the parent of the vertex number *y*), *w* and *p* are the weight and the strength of the edge, accordingly. It is guaranteed that the edges describe the tree with the root in the vertex 1.

**Output**

If it is impossible to get unbroken tree from the given tree, print -1 in the only line.

Otherwise, the output data should contain *n* lines:

In the first line print the number *n* — the number of vertices on the tree.

In the next *n* - 1 lines print the description of edges of the resulting tree. Each line should contain four integers *x*, *y*, *w*, *p*(1 ≤ *x*, *y* ≤ *n*, 1 ≤ *w* ≤ 109, 0 ≤ *p* ≤ 109), where *x* and *y* — vertices, which the edge connects (the vertex number *x* is the parent of the vertex number *y*), *w* and *p* are the new weight and the strength of the edge, accordingly.

Print edges in the same order as they are given in input data: the first two integers of each line should not be changed.

**Examples**

**input**

3  
1 3 5 7  
3 2 4 3

**output**

3  
1 3 5 7  
3 2 4 3

**input**

4  
1 3 2 3  
3 4 5 1  
3 2 3 3

**output**

-1

**input**

5  
1 2 2 4  
2 4 1 9  
4 5 5 6  
4 3 4 8

**output**

5  
1 2 2 4  
2 4 1 9  
4 5 1 2  
4 3 2 6

**input**

7  
1 2 5 2  
2 3 4 3  
1 4 3 7  
4 5 4 1  
4 6 3 2  
6 7 1 6

**output**

7  
1 2 5 2  
2 3 2 1  
1 4 3 7  
4 5 3 0  
4 6 3 2  
6 7 1 6

D. Ability To Convert

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Alexander is learning how to convert numbers from the decimal system to any other, however, he doesn't know English letters, so he writes any number only as a decimal number, it means that instead of the letter *A* he will write the number 10. Thus, by converting the number 475 from decimal to hexadecimal system, he gets 11311 (475 = 1·162 + 13·161 + 11·160). Alexander lived calmly until he tried to convert the number back to the decimal number system.

Alexander remembers that he worked with little numbers so he asks to find the minimum decimal number so that by converting it to the system with the base *n* he will get the number *k*.

**Input**

The first line contains the integer *n* (2 ≤ *n* ≤ 109). The second line contains the integer *k* (0 ≤ *k* < 1060), it is guaranteed that the number *k* contains no more than 60 symbols. All digits in the second line are strictly less than *n*.

Alexander guarantees that the answer exists and does not exceed 1018.

The number *k* doesn't contain leading zeros.

**Output**

Print the number *x* (0 ≤ *x* ≤ 1018) — the answer to the problem.

**Examples**

**input**

13  
12

**output**

12

**input**

16  
11311

**output**

475

**input**

20  
999

**output**

3789

**input**

17  
2016

**output**

594

**Note**

In the first example 12 could be obtained by converting two numbers to the system with base 13: 12 = 12·130 or 15 = 1·131 + 2·130.

C. Unfair Poll

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

On the Literature lesson Sergei noticed an awful injustice, it seems that some students are asked more often than others.

Seating in the class looks like a rectangle, where *n* rows with *m* pupils in each.

The teacher asks pupils in the following order: at first, she asks all pupils from the first row in the order of their seating, then she continues to ask pupils from the next row. If the teacher asked the last row, then the direction of the poll changes, it means that she asks the previous row. The order of asking the rows looks as follows: the 1-st row, the 2-nd row, ..., the *n* - 1-st row, the *n*-th row, the *n* - 1-st row, ..., the 2-nd row, the 1-st row, the 2-nd row, ...

The order of asking of pupils on the same row is always the same: the 1-st pupil, the 2-nd pupil, ..., the *m*-th pupil.

During the lesson the teacher managed to ask exactly *k* questions from pupils in order described above. Sergei seats on the *x*-th row, on the *y*-th place in the row. Sergei decided to prove to the teacher that pupils are asked irregularly, help him count three values:

1. the maximum number of questions a particular pupil is asked,
2. the minimum number of questions a particular pupil is asked,
3. how many times the teacher asked Sergei.

If there is only one row in the class, then the teacher always asks children from this row.

**Input**

The first and the only line contains five integers *n*, *m*, *k*, *x* and *y* (1 ≤ *n*, *m* ≤ 100, 1 ≤ *k* ≤ 1018, 1 ≤ *x* ≤ *n*, 1 ≤ *y* ≤ *m*).

**Output**

Print three integers:

1. the maximum number of questions a particular pupil is asked,
2. the minimum number of questions a particular pupil is asked,
3. how many times the teacher asked Sergei.

**Examples**

**input**

1 3 8 1 1

**output**

3 2 3

**input**

4 2 9 4 2

**output**

2 1 1

**input**

5 5 25 4 3

**output**

1 1 1

**input**

100 100 1000000000000000000 100 100

**output**

101010101010101 50505050505051 50505050505051

**Note**

The order of asking pupils in the first test:

1. the pupil from the first row who seats at the first table, it means it is Sergei;
2. the pupil from the first row who seats at the second table;
3. the pupil from the first row who seats at the third table;
4. the pupil from the first row who seats at the first table, it means it is Sergei;
5. the pupil from the first row who seats at the second table;
6. the pupil from the first row who seats at the third table;
7. the pupil from the first row who seats at the first table, it means it is Sergei;
8. the pupil from the first row who seats at the second table;

The order of asking pupils in the second test:

1. the pupil from the first row who seats at the first table;
2. the pupil from the first row who seats at the second table;
3. the pupil from the second row who seats at the first table;
4. the pupil from the second row who seats at the second table;
5. the pupil from the third row who seats at the first table;
6. the pupil from the third row who seats at the second table;
7. the pupil from the fourth row who seats at the first table;
8. the pupil from the fourth row who seats at the second table, it means it is Sergei;
9. the pupil from the third row who seats at the first table;

B. Blown Garland

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Nothing is eternal in the world, Kostya understood it on the 7-th of January when he saw partially dead four-color garland.

Now he has a goal to replace dead light bulbs, however he doesn't know how many light bulbs for each color are required. It is guaranteed that for each of four colors at least one light is working.

It is known that the garland contains light bulbs of four colors: red, blue, yellow and green. The garland is made as follows: if you take any four consecutive light bulbs then there will not be light bulbs with the same color among them. For example, the garland can look like "RYBGRYBGRY", "YBGRYBGRYBG", "BGRYB", but can not look like "BGRYG", "YBGRYBYGR" or "BGYBGY". Letters denote colors: 'R' — red, 'B' — blue, 'Y' — yellow, 'G' — green.

Using the information that for each color at least one light bulb still works count the number of dead light bulbs of each four colors.

**Input**

The first and the only line contains the string *s* (4 ≤ |*s*| ≤ 100), which describes the garland, the *i*-th symbol of which describes the color of the *i*-th light bulb in the order from the beginning of garland:

* 'R' — the light bulb is red,
* 'B' — the light bulb is blue,
* 'Y' — the light bulb is yellow,
* 'G' — the light bulb is green,
* '!' — the light bulb is dead.

The string *s* can not contain other symbols except those five which were described.

It is guaranteed that in the given string at least once there is each of four letters 'R', 'B', 'Y' and 'G'.

It is guaranteed that the string *s* is correct garland with some blown light bulbs, it means that for example the line "GRBY!!!B" can not be in the input data.

**Output**

In the only line print four integers *kr*, *kb*, *ky*, *kg* — the number of dead light bulbs of red, blue, yellow and green colors accordingly.

**Examples**

**input**

RYBGRYBGR

**output**

0 0 0 0

**input**

!RGYB

**output**

0 1 0 0

**input**

!!!!YGRB

**output**

1 1 1 1

**input**

!GB!RG!Y!

**output**

2 1 1 0

**Note**

In the first example there are no dead light bulbs.

In the second example it is obvious that one blue bulb is blown, because it could not be light bulbs of other colors on its place according to the statements.

A. Holiday Of Equality

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

In Berland it is the holiday of equality. In honor of the holiday the king decided to equalize the welfare of all citizens in Berland by the expense of the state treasury.

Totally in Berland there are *n* citizens, the welfare of each of them is estimated as the integer in *ai* burles (burle is the currency in Berland).

You are the royal treasurer, which needs to count the minimum charges of the kingdom on the king's present. The king can only give money, he hasn't a power to take away them.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 100) — the number of citizens in the kingdom.

The second line contains *n* integers *a*1, *a*2, ..., *an*, where *ai* (0 ≤ *ai* ≤ 106) — the welfare of the *i*-th citizen.

**Output**

In the only line print the integer *S* — the minimum number of burles which are had to spend.

**Examples**

**input**

5  
0 1 2 3 4

**output**

10

**input**

5  
1 1 0 1 1

**output**

1

**input**

3  
1 3 1

**output**

4

**input**

1  
12

**output**

0

**Note**

In the first example if we add to the first citizen 4 burles, to the second 3, to the third 2 and to the fourth 1, then the welfare of all citizens will equal 4.

In the second example it is enough to give one burle to the third citizen.

In the third example it is necessary to give two burles to the first and the third citizens to make the welfare of citizens equal 3.

In the fourth example it is possible to give nothing to everyone because all citizens have 12 burles.

G. Can Bash Save the Day?

time limit per test

5 seconds

memory limit per test

768 megabytes

input

standard input

output

standard output

Whoa! You did a great job helping Team Rocket who managed to capture all the Pokemons sent by Bash. Meowth, part of Team Rocket, having already mastered the human language, now wants to become a master in programming as well. He agrees to free the Pokemons if Bash can answer his questions.

Initially, Meowth gives Bash a weighted tree containing *n* nodes and a sequence *a*1, *a*2..., *an* which is a permutation of 1, 2, ..., *n*. Now, Mewoth makes *q* queries of one of the following forms:

* 1 l r v: meaning Bash should report http://codeforces.com/predownloaded/d5/12/d512c61396fef6ee39c288ac2b19a129e12fb552.png, where *dist*(*a*, *b*) is the length of the shortest path from node *a* to node *b* in the given tree.
* 2 x: meaning Bash should swap *ax* and *ax*+ 1 in the given sequence. This new sequence is used for later queries.

Help Bash to answer the questions!

**Input**

The first line contains two integers *n* and *q* (1 ≤ *n* ≤ 2·105, 1 ≤ *q* ≤ 2·105) — the number of nodes in the tree and the number of queries, respectively.

The next line contains *n* space-separated integers — the sequence *a*1, *a*2, ..., *an* which is a permutation of 1, 2, ..., *n*.

Each of the next *n* - 1 lines contain three space-separated integers *u*, *v*, and *w* denoting that there exists an undirected edge between node *u* and node *v* of weight *w*, (1 ≤ *u*, *v* ≤ *n*, *u* ≠ *v*, 1 ≤ *w* ≤ 106). It is guaranteed that the given graph is a tree.

Each query consists of two lines. First line contains single integer *t*, indicating the type of the query. Next line contains the description of the query:

* **t = 1**: Second line contains three integers *a*, *b* and *c* (1 ≤ *a*, *b*, *c* < 230) using which *l*, *r* and *v* can be generated using the formula given below:
  + http://codeforces.com/predownloaded/46/7e/467e7bac706913a637ae6a4c27a622d820bc8a3b.png,
  + http://codeforces.com/predownloaded/5e/69/5e69ee3de7773d36423df01c1f9ed59c0898a6ac.png,
  + http://codeforces.com/predownloaded/d1/e7/d1e70d9053ae70f06241d3e49026ca2c0dc48611.png.
* **t = 2**: Second line contains single integer *a* (1 ≤ *a* < 230) using which *x* can be generated using the formula given below:
  + http://codeforces.com/predownloaded/49/bd/49bd28ebf3bf2859f9ddcd0b6d16fb94d73cc3c4.png.

The *ansi* is the answer for the *i*-th query, assume that *ans*0 = 0. If the *i*-th query is of type 2 then *ansi* = *ansi*- 1. It is guaranteed that:

* **for each query of type 1**: 1 ≤ *l* ≤ *r* ≤ *n*, 1 ≤ *v* ≤ *n*,
* **for each query of type 2**: 1 ≤ *x* ≤ *n* - 1.

The http://codeforces.com/predownloaded/7b/ea/7beade55e90846d70020a3d03521d3458b66751b.png operation means bitwise exclusive OR.

**Output**

For each query of type 1, output a single integer in a separate line, denoting the answer to the query.

**Example**

**input**

5 5  
4 5 1 3 2  
4 2 4  
1 3 9  
4 1 4  
4 5 2  
1  
1 5 4  
1  
22 20 20  
2  
38  
2  
39  
1  
36 38 38

**output**

23  
37  
28

**Note**

In the sample, the actual queries are the following:

* 1 1 5 4
* 1 1 3 3
* 2 3
* 2 2
* 1 1 3 3

F. Team Rocket Rises Again

time limit per test

2.5 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

It's the turn of the year, so Bash wants to send presents to his friends. There are *n* cities in the Himalayan region and they are connected by *m* bidirectional roads. Bash is living in city *s*. Bash has exactly one friend in each of the other cities. Since Bash wants to surprise his friends, he decides to send a Pikachu to each of them. **Since there may be some cities which are not reachable from Bash's city, he only sends a Pikachu to those friends who live in a city reachable from his own city**. He also wants to send it to them as soon as possible.

He finds out the minimum time for each of his Pikachus to reach its destination city. Since he is a perfectionist, he informs all his friends with the time their gift will reach them. A Pikachu travels at a speed of 1 meters per second. His friends were excited to hear this and would be unhappy if their presents got delayed. Unfortunately Team Rocket is on the loose and they came to know of Bash's plan. They want to maximize the number of friends who are unhappy with Bash.

They do this by destroying exactly one of the other *n* - 1 cities. This implies that **the friend residing in that city dies, so he is unhappy as well**.

Note that **if a city is destroyed, all the roads directly connected to the city are also destroyed and the Pikachu may be forced to take a longer alternate route**.

**Please also note that only friends that are waiting for a gift count as unhappy, even if they die.**

Since Bash is already a legend, can you help Team Rocket this time and find out the maximum number of Bash's friends who can be made unhappy by destroying exactly one city.

**Input**

The first line contains three space separated integers *n*, *m* and *s* (2 ≤ *n* ≤ 2·105, http://codeforces.com/predownloaded/d3/e1/d3e1c0f42d504617a92d8ab9b74407665584a7b0.png, 1 ≤ *s* ≤ *n*) — the number of cities and the number of roads in the Himalayan region and the city Bash lives in.

Each of the next *m* lines contain three space-separated integers *u*, *v* and *w* (1 ≤ *u*, *v* ≤ *n*, *u* ≠ *v*, 1 ≤ *w* ≤ 109) denoting that there exists a road between city *u* and city *v* of length *w* meters.

It is guaranteed that no road connects a city to itself and there are no two roads that connect the same pair of cities.

**Output**

Print a single integer, the answer to the problem.

**Examples**

**input**

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

**output**

2

**input**

7 11 2  
1 2 5  
1 3 5  
2 4 2  
2 5 2  
3 6 3  
3 7 3  
4 6 2  
3 4 2  
6 7 3  
4 5 7  
4 7 7

**output**

4

**Note**

In the first sample, on destroying the city 2, the length of shortest distance between pairs of cities (3, 2) and (3, 4) will change. Hence the answer is 2.

E. Bash Plays with Functions

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Bash got tired on his journey to become the greatest Pokemon master. So he decides to take a break and play with functions.

Bash defines a function *f*0(*n*), which denotes the number of ways of factoring *n* into two factors *p* and *q* such that *gcd*(*p*, *q*) = 1. In other words, *f*0(*n*) is the number of ordered pairs of positive integers (*p*, *q*) such that *p*·*q* = *n* and *gcd*(*p*, *q*) = 1.

But Bash felt that it was too easy to calculate this function. So he defined a series of functions, where *fr*+ 1 is defined as:

http://codeforces.com/predownloaded/e7/e1/e7e119e8b4161c17fb9b8c8068d7edfe4390410d.png

Where (*u*, *v*) is any ordered pair of positive integers, they need not to be co-prime.

Now Bash wants to know the value of *fr*(*n*) for different *r* and *n*. Since the value could be huge, he would like to know the value modulo 109 + 7. Help him!

**Input**

The first line contains an integer *q* (1 ≤ *q* ≤ 106) — the number of values Bash wants to know.

Each of the next *q* lines contain two integers *r* and *n* (0 ≤ *r* ≤ 106, 1 ≤ *n* ≤ 106), which denote Bash wants to know the value *fr*(*n*).

**Output**

Print *q* integers. For each pair of *r* and *n* given, print *fr*(*n*) modulo 109 + 7 on a separate line.

**Example**

**input**

5  
0 30  
1 25  
3 65  
2 5  
4 48

**output**

8  
5  
25  
4  
630

D. Felicity's Big Secret Revealed

time limit per test

4 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

The gym leaders were fascinated by the evolutions which took place at Felicity camp. So, they were curious to know about the secret behind evolving Pokemon.

The organizers of the camp gave the gym leaders a PokeBlock, a sequence of *n* ingredients. Each ingredient can be of type 0 or 1. Now the organizers told the gym leaders that to evolve a Pokemon of type *k* (*k* ≥ 2), they need to make a valid set of *k* cuts on the PokeBlock to get smaller blocks.

Suppose the given PokeBlock sequence is *b*0*b*1*b*2... *bn*- 1. You have a choice of making cuts at *n* + 1 places, i.e., Before *b*0, between *b*0 and *b*1, between *b*1 and *b*2, ..., between *bn*- 2 and *bn*- 1, and after *bn*- 1.

The *n* + 1 choices of making cuts are as follows (where a | denotes a possible cut):

| *b*0 | *b*1 | *b*2 | ... | *bn*- 2 | *bn*- 1 |

Consider a sequence of *k* cuts. Now each pair of consecutive cuts will contain a binary string between them, formed from the ingredient types. The ingredients before the first cut and after the last cut are wasted, which is to say they are not considered. So there will be exactly *k* - 1 such binary substrings. Every substring can be read as a binary number. Let *m* be the maximum number out of the obtained numbers. If all the obtained numbers are positive and the set of the obtained numbers contains all integers from 1 to *m*, then this set of cuts is said to be a valid set of cuts.

For example, suppose the given PokeBlock sequence is 101101001110 and we made 5 cuts in the following way:

10 | 11 | 010 | 01 | 1 | 10

So the 4 binary substrings obtained are: 11, 010, 01 and 1, which correspond to the numbers 3, 2, 1 and 1 respectively. Here *m* = 3, as it is the maximum value among the obtained numbers. And all the obtained numbers are positive and we have obtained all integers from 1 to *m*. Hence this set of cuts is a valid set of 5 cuts.

A Pokemon of type *k* will evolve only if the PokeBlock is cut using a valid set of *k* cuts. There can be many valid sets of the same size. Two valid sets of *k* cuts are considered different if there is a cut in one set which is not there in the other set.

Let *f*(*k*) denote the number of valid sets of *k* cuts. Find the value of http://codeforces.com/predownloaded/ed/48/ed481fee98701621da5df39e7f9a10884f9b5cde.png. Since the value of *s* can be very large, output *s*modulo 109 + 7.

**Input**

The input consists of two lines. The first line consists an integer *n* (1 ≤ *n* ≤ 75) — the length of the PokeBlock. The next line contains the PokeBlock, a binary string of length *n*.

**Output**

Output a single integer, containing the answer to the problem, i.e., the value of *s* modulo 109 + 7.

**Examples**

**input**

4  
1011

**output**

10

**input**

2  
10

**output**

1

**Note**

In the first sample, the sets of valid cuts are:

Size 2: |1|011, 1|01|1, 10|1|1, 101|1|.

Size 3: |1|01|1, |10|1|1, 10|1|1|, 1|01|1|.

Size 4: |10|1|1|, |1|01|1|.

Hence, *f*(2) = 4, *f*(3) = 4 and *f*(4) = 2. So, the value of *s* = 10.

In the second sample, the set of valid cuts is:

Size 2: |1|0.

Hence, *f*(2) = 1 and *f*(3) = 0. So, the value of *s* = 1.

C. Felicity is Coming!

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

It's that time of the year, Felicity is around the corner and you can see people celebrating all around the Himalayan region. The Himalayan region has *n* gyms. The *i*-th gym has *gi* Pokemon in it. There are *m* distinct Pokemon types in the Himalayan region numbered from 1 to *m*. There is a special evolution camp set up in the fest which claims to evolve any Pokemon. The type of a Pokemon could change after evolving, subject to the constraint that if two Pokemon have the same type before evolving, they will have the same type after evolving. Also, if two Pokemon have different types before evolving, they will have different types after evolving. It is also possible that a Pokemon has the same type before and after evolving.

Formally, an *evolution plan* is a permutation *f* of {1, 2, ..., *m*}, such that *f*(*x*) = *y* means that a Pokemon of type *x* evolves into a Pokemon of type *y*.

The gym leaders are intrigued by the special evolution camp and all of them plan to evolve their Pokemons. The protocol of the mountain states that in each gym, for every type of Pokemon, the number of Pokemon of that type before evolving any Pokemon should be equal the number of Pokemon of that type after evolving all the Pokemons according to the evolution plan. They now want to find out how many distinct *evolution plans* exist which satisfy the protocol.

Two evolution plans *f*1 and *f*2 are distinct, if they have at least one Pokemon type evolving into a different Pokemon type in the two plans, i. e. there exists an *i* such that *f*1(*i*) ≠ *f*2(*i*).

Your task is to find how many distinct *evolution plans* are possible such that if all Pokemon in all the gyms are evolved, the number of Pokemon of each type in each of the gyms remains the same. As the answer can be large, output it modulo 109 + 7.

**Input**

The first line contains two integers *n* and *m* (1 ≤ *n* ≤ 105, 1 ≤ *m* ≤ 106) — the number of gyms and the number of Pokemon types.

The next *n* lines contain the description of Pokemons in the gyms. The *i*-th of these lines begins with the integer *gi* (1 ≤ *gi* ≤ 105) — the number of Pokemon in the *i*-th gym. After that *gi* integers follow, denoting types of the Pokemons in the *i*-th gym. Each of these integers is between 1 and *m*.

The total number of Pokemons (the sum of all *gi*) does not exceed 5·105.

**Output**

Output the number of valid evolution plans modulo 109 + 7.

**Examples**

**input**

2 3  
2 1 2  
2 2 3

**output**

1

**input**

1 3  
3 1 2 3

**output**

6

**input**

2 4  
2 1 2  
3 2 3 4

**output**

2

**input**

2 2  
3 2 2 1  
2 1 2

**output**

1

**input**

3 7  
2 1 2  
2 3 4  
3 5 6 7

**output**

24

**Note**

In the first case, the only possible evolution plan is:

http://codeforces.com/predownloaded/40/36/40363443841f1b03ec9f07abfc4a960efce965e6.png

In the second case, any permutation of (1,  2,  3) is valid.

In the third case, there are two possible plans:

http://codeforces.com/predownloaded/d3/98/d39891c84bde81066c2b8702b5608fa4cb1d97a3.png

http://codeforces.com/predownloaded/d8/fc/d8fcb408b6b29f2b48fb4b64cfaee9a196a77459.png

In the fourth case, the only possible evolution plan is:

http://codeforces.com/predownloaded/f6/ef/f6ef0fb3e17f35b18b4e785400b9fee584093527.png

B. Bash's Big Day

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Bash has set out on a journey to become the greatest Pokemon master. To get his first Pokemon, he went to Professor Zulu's Lab. Since Bash is Professor Zulu's favourite student, Zulu allows him to take as many Pokemon from his lab as he pleases.

But Zulu warns him that a group of *k* > 1 Pokemon with strengths {*s*1, *s*2, *s*3, ..., *sk*} tend to fight among each other if *gcd*(*s*1, *s*2, *s*3, ..., *sk*) = 1 (see notes for *gcd* definition).

Bash, being smart, does not want his Pokemon to fight among each other. However, he also wants to maximize the number of Pokemon he takes from the lab. Can you help Bash find out the maximum number of Pokemon he can take?

**Note**: A Pokemon cannot fight with itself.

**Input**

The input consists of two lines.

The first line contains an integer *n* (1 ≤ *n* ≤ 105), the number of Pokemon in the lab.

The next line contains *n* space separated integers, where the *i*-th of them denotes *si* (1 ≤ *si* ≤ 105), the strength of the *i*-th Pokemon.

**Output**

Print single integer — the maximum number of Pokemons Bash can take.

**Examples**

**input**

3  
2 3 4

**output**

2

**input**

5  
2 3 4 6 7

**output**

3

**Note**

*gcd* (greatest common divisor) of positive integers set {*a*1, *a*2, ..., *an*} is the maximum positive integer that divides all the integers {*a*1, *a*2, ..., *an*}.

In the first sample, we can take Pokemons with strengths {2, 4} since *gcd*(2, 4) = 2.

In the second sample, we can take Pokemons with strengths {2, 4, 6}, and there is no larger group with *gcd* ≠ 1.

A. Gotta Catch Em' All!

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Bash wants to become a Pokemon master one day. Although he liked a lot of Pokemon, he has always been fascinated by Bulbasaur the most. Soon, things started getting serious and his fascination turned into an obsession. Since he is too young to go out and catch Bulbasaur, he came up with his own way of catching a Bulbasaur.

Each day, he takes the front page of the newspaper. He cuts out the letters one at a time, from anywhere on the front page of the newspaper to form the word "Bulbasaur" (without quotes) and sticks it on his wall. Bash is very particular about case — the first letter of "Bulbasaur" must be upper case and the rest must be lower case. By doing this he thinks he has caught one Bulbasaur. He then repeats this step on the left over part of the newspaper. He keeps doing this until it is not possible to form the word "Bulbasaur" from the newspaper.

Given the text on the front page of the newspaper, can you tell how many Bulbasaurs he will catch today?

Note: **uppercase and lowercase letters are considered different.**

**Input**

Input contains a single line containing a string *s* (1  ≤  |*s*|  ≤  105) — the text on the front page of the newspaper without spaces and punctuation marks. |*s*| is the length of the string *s*.

The string *s* contains lowercase and uppercase English letters, i.e. http://codeforces.com/predownloaded/a0/53/a0535434e215fbcfeb548858f5accaa96f311b13.png.

**Output**

Output a single integer, the answer to the problem.

**Examples**

**input**

Bulbbasaur

**output**

1

**input**

F

**output**

0

**input**

aBddulbasaurrgndgbualdBdsagaurrgndbb

**output**

2

**Note**

In the first case, you could pick: **Bulb**b**asaur**.

In the second case, there is no way to pick even a single Bulbasaur.

In the third case, you can rearrange the string to **BulbasaurBulbasaur**addrgndgddgargndbb to get two words "Bulbasaur".

F. Long number

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Consider the following grammar:

* <expression> ::= <term> | <expression> '+' <term>
* <term> ::= <number> | <number> '-' <number> | <number> '(' <expression> ')'
* <number> ::= <pos\_digit> | <number> <digit>
* <digit> ::= '0' | <pos\_digit>
* <pos\_digit> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

This grammar describes a number in decimal system using the following rules:

* <number> describes itself,
* <number>-<number> (l-r, *l* ≤ *r*) describes integer which is concatenation of all integers from *l* to *r*, written without leading zeros. For example, 8-11 describes 891011,
* <number>(<expression>) describes integer which is concatenation of <number> copies of integer described by <expression>,
* <expression>+<term> describes integer which is concatenation of integers described by <expression> and <term>.

For example, 2(2-4+1)+2(2(17)) describes the integer 2341234117171717.

You are given an expression in the given grammar. Print the integer described by it modulo 109 + 7.

**Input**

The only line contains a non-empty string at most 105 characters long which is valid according to the given grammar. In particular, it means that in terms l-r *l* ≤ *r* holds.

**Output**

Print single integer — the number described by the expression modulo 109 + 7.

**Examples**

**input**

8-11

**output**

891011

**input**

2(2-4+1)+2(2(17))

**output**

100783079

**input**

1234-5678

**output**

745428774

**input**

1+2+3+4-5+6+7-9

**output**

123456789

E. Byteland coins

time limit per test

1 second

memory limit per test

512 megabytes

input

standard input

output

standard output

There are *n* types of coins in Byteland. Conveniently, the denomination of the coin type *k* divides the denomination of the coin type *k* + 1, the denomination of the coin type 1 equals 1 tugrick. The ratio of the denominations of coin types *k* + 1 and *k* equals *ak*. It is known that for each *x* there are at most **20** coin types of denomination *x*.

Byteasar has *bk* coins of type *k* with him, and he needs to pay exactly *m* tugricks. It is known that Byteasar never has more than 3·105coins with him. Byteasar want to know how many ways there are to pay exactly *m* tugricks. Two ways are different if there is an integer *k*such that the amount of coins of type *k* differs in these two ways. As all Byteland citizens, Byteasar wants to know the number of ways modulo 109 + 7.

**Input**

The first line contains single integer *n* (1 ≤ *n* ≤ 3·105) — the number of coin types.

The second line contains *n* - 1 integers *a*1, *a*2, ..., *an*- 1 (1 ≤ *ak* ≤ 109) — the ratios between the coin types denominations. It is guaranteed that for each *x* there are at most **20** coin types of denomination *x*.

The third line contains *n* non-negative integers *b*1, *b*2, ..., *bn* — the number of coins of each type Byteasar has. It is guaranteed that the sum of these integers doesn't exceed 3·105.

The fourth line contains single integer *m* (0 ≤ *m* < 1010000) — the amount in tugricks Byteasar needs to pay.

**Output**

Print single integer — the number of ways to pay exactly *m* tugricks modulo 109 + 7.

**Examples**

**input**

1  
  
4  
2

**output**

1

**input**

2  
1  
4 4  
2

**output**

3

**input**

3  
3 3  
10 10 10  
17

**output**

6

**Note**

In the first example Byteasar has 4 coins of denomination 1, and he has to pay 2 tugricks. There is only one way.

In the second example Byteasar has 4 coins of each of two different types of denomination 1, he has to pay 2 tugricks. There are 3ways: pay one coin of the first type and one coin of the other, pay two coins of the first type, and pay two coins of the second type.

In the third example the denominations are equal to 1, 3, 9.

D. Bacterial Melee

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Julia is conducting an experiment in her lab. She placed several luminescent bacterial colonies in a horizontal testtube. Different types of bacteria can be distinguished by the color of light they emit. Julia marks types of bacteria with small Latin letters "a", ..., "z".

The testtube is divided into *n* consecutive regions. Each region is occupied by a single colony of a certain bacteria type at any given moment. Hence, the population of the testtube at any moment can be described by a string of *n* Latin characters.

Sometimes a colony can decide to conquer another colony in one of the adjacent regions. When that happens, the attacked colony is immediately eliminated and replaced by a colony of the same type as the attacking colony, while the attacking colony keeps its type. Note that a colony can only attack its neighbours within the boundaries of the testtube. At any moment, at most one attack can take place.

For example, consider a testtube with population "babb". There are six options for an attack that may happen next:

* the first colony attacks the second colony (1 → 2), the resulting population is "bbbb";
* 2 → 1, the result is "aabb";
* 2 → 3, the result is "baab";
* 3 → 2, the result is "bbbb" (note that the result is the same as the first option);
* 3 → 4 or 4 → 3, the population does not change.

The pattern of attacks is rather unpredictable. Julia is now wondering how many different configurations of bacteria in the testtube she can obtain after a sequence of attacks takes place (it is possible that no attacks will happen at all). Since this number can be large, find it modulo 109 + 7.

**Input**

The first line contains an integer *n* — the number of regions in the testtube (1 ≤ *n* ≤ 5 000).

The second line contains *n* small Latin letters that describe the initial population of the testtube.

**Output**

Print one number — the answer to the problem modulo 109 + 7.

**Examples**

**input**

3  
aaa

**output**

1

**input**

2  
ab

**output**

3

**input**

4  
babb

**output**

11

**input**

7  
abacaba

**output**

589

**Note**

In the first sample the population can never change since all bacteria are of the same type.

In the second sample three configurations are possible: "ab" (no attacks), "aa" (the first colony conquers the second colony), and "bb" (the second colony conquers the first colony).

To get the answer for the third sample, note that more than one attack can happen.

C. Nikita and stack

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Nikita has a stack. A stack in this problem is a data structure that supports two operations. Operation push(x) puts an integer *x* on the top of the stack, and operation pop() deletes the top integer from the stack, i. e. the last added. If the stack is empty, then the operation pop() does nothing.

Nikita made *m* operations with the stack but forgot them. Now Nikita wants to remember them. He remembers them one by one, on the *i*-th step he remembers an operation he made *pi*-th. In other words, he remembers the operations in order of some permutation *p*1, *p*2, ..., *pm*. After each step Nikita wants to know what is the integer on the top of the stack after performing the operations he have already remembered, in the corresponding order. Help him!

**Input**

The first line contains the integer *m* (1 ≤ *m* ≤ 105) — the number of operations Nikita made.

The next *m* lines contain the operations Nikita remembers. The *i*-th line starts with two integers *pi* and *ti* (1 ≤ *pi* ≤ *m*, *ti* = 0 or *ti* = 1) — the index of operation he remembers on the step *i*, and the type of the operation. *ti* equals 0, if the operation is pop(), and 1, is the operation is push(x). If the operation is push(x), the line also contains the integer *xi* (1 ≤ *xi* ≤ 106) — the integer added to the stack.

It is guaranteed that each integer from 1 to *m* is present exactly once among integers *pi*.

**Output**

Print *m* integers. The integer *i* should equal the number on the top of the stack after performing all the operations Nikita remembered on the steps from 1 to *i*. If the stack is empty after performing all these operations, print -1.

**Examples**

**input**

2  
2 1 2  
1 0

**output**

2  
2

**input**

3  
1 1 2  
2 1 3  
3 0

**output**

2  
3  
2

**input**

5  
5 0  
4 0  
3 1 1  
2 1 1  
1 1 2

**output**

-1  
-1  
-1  
-1  
2

**Note**

In the first example, after Nikita remembers the operation on the first step, the operation push(2) is the only operation, so the answer is 2. After he remembers the operation pop() which was done before push(2), answer stays the same.

In the second example, the operations are push(2), push(3) and pop(). Nikita remembers them in the order they were performed.

In the third example Nikita remembers the operations in the reversed order.

B. Travel Card

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

A new innovative ticketing systems for public transport is introduced in Bytesburg. Now there is a single travel card for all transport. To make a trip a passenger scan his card and then he is charged according to the fare.

The fare is constructed in the following manner. There are three types of tickets:

1. a ticket for one trip costs 20 byteland rubles,
2. a ticket for 90 minutes costs 50 byteland rubles,
3. a ticket for one day (1440 minutes) costs 120 byteland rubles.

Note that a ticket for *x* minutes activated at time *t* can be used for trips started in time range from *t* to *t* + *x* - 1, inclusive. Assume that all trips take exactly one minute.

To simplify the choice for the passenger, the system automatically chooses the optimal tickets. After each trip starts, the system analyses all the previous trips and the current trip and chooses a set of tickets for these trips with a minimum total cost. Let the minimum total cost of tickets to cover all trips from the first to the current is *a*, and the total sum charged before is *b*. Then the system charges the passenger the sum *a* - *b*.

You have to write a program that, for given trips made by a passenger, calculates the sum the passenger is charged after each trip.

**Input**

The first line of input contains integer number *n* (1 ≤ *n* ≤ 105) — the number of trips made by passenger.

Each of the following *n* lines contains the time of trip *ti* (0 ≤ *ti* ≤ 109), measured in minutes from the time of starting the system. All *ti* are different, given in ascending order, i. e. *ti*+ 1 > *ti* holds for all 1 ≤ *i* < *n*.

**Output**

Output *n* integers. For each trip, print the sum the passenger is charged after it.

**Examples**

**input**

3  
10  
20  
30

**output**

20  
20  
10

**input**

10  
13  
45  
46  
60  
103  
115  
126  
150  
256  
516

**output**

20  
20  
10  
0  
20  
0  
0  
20  
20  
10

**Note**

In the first example, the system works as follows: for the first and second trips it is cheaper to pay for two one-trip tickets, so each time 20 rubles is charged, after the third trip the system understands that it would be cheaper to buy a ticket for 90 minutes. This ticket costs 50 rubles, and the passenger had already paid 40 rubles, so it is necessary to charge 10 rubles only.

A. Pavel and barbecue

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Pavel cooks barbecue. There are *n* skewers, they lay on a brazier in a row, each on one of *n* positions. Pavel wants each skewer to be cooked some time in every of *n* positions in two directions: in the one it was directed originally and in the reversed direction.

Pavel has a plan: a permutation *p* and a sequence *b*1, *b*2, ..., *bn*, consisting of zeros and ones. Each second Pavel move skewer on position *i* to position *pi*, and if *bi* equals 1 then he reverses it. So he hope that every skewer will visit every position in both directions.

Unfortunately, not every pair of permutation *p* and sequence *b* suits Pavel. What is the minimum total number of elements in the given permutation *p* and the given sequence *b* he needs to change so that every skewer will visit each of 2*n* placements? Note that after changing the permutation should remain a permutation as well.

There is no problem for Pavel, if some skewer visits some of the placements several times before he ends to cook. In other words, a permutation *p* and a sequence *b* suit him if there is an integer *k* (*k* ≥ 2*n*), so that after *k* seconds each skewer visits each of the 2*n*placements.

It can be shown that some suitable pair of permutation *p* and sequence *b* exists for any *n*.

**Input**

The first line contain the integer *n* (1 ≤ *n* ≤ 2·105) — the number of skewers.

The second line contains a sequence of integers *p*1, *p*2, ..., *pn* (1 ≤ *pi* ≤ *n*) — the permutation, according to which Pavel wants to move the skewers.

The third line contains a sequence *b*1, *b*2, ..., *bn* consisting of zeros and ones, according to which Pavel wants to reverse the skewers.

**Output**

Print single integer — the minimum total number of elements in the given permutation *p* and the given sequence *b* he needs to change so that every skewer will visit each of 2*n* placements.

**Examples**

**input**

4  
4 3 2 1  
0 1 1 1

**output**

2

**input**

3  
2 3 1  
0 0 0

**output**

1

**Note**

In the first example Pavel can change the permutation to 4, 3, 1, 2.

In the second example Pavel can change any element of *b* to 1.

G. PolandBall and Many Other Balls

time limit per test

6 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

PolandBall is standing in a row with Many Other Balls. More precisely, there are exactly *n* Balls. Balls are proud of their home land — and they want to prove that it's strong.

The Balls decided to start with selecting exactly *m* groups of Balls, each consisting either of single Ball or two neighboring Balls. Each Ball can join no more than one group.

The Balls really want to impress their Enemies. They kindly asked you to calculate number of such divisions for all *m* where 1 ≤ *m* ≤ *k*. Output all these values modulo 998244353, the Enemies will be impressed anyway.

**Input**

There are exactly two numbers *n* and *k* (1 ≤ *n* ≤ 109, 1 ≤ *k* < 215), denoting the number of Balls and the maximim number of groups, respectively.

**Output**

You should output a sequence of *k* values. The *i*-th of them should represent the sought number of divisions into exactly *i* groups, according to PolandBall's rules.

**Examples**

**input**

3 3

**output**

5 5 1

**input**

1 1

**output**

1

**input**

5 10

**output**

9 25 25 9 1 0 0 0 0 0

**Note**

In the first sample case we can divide Balls into groups as follows:

{1}, {2}, {3}, {12}, {23}.

{12}{3}, {1}{23}, {1}{2}, {1}{3}, {2}{3}.

{1}{2}{3}.

Therefore, output is: 5 5 1.

F. PolandBall and Gifts

time limit per test

1.5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

It's Christmas time! PolandBall and his friends will be giving themselves gifts. There are *n* Balls overall. Each Ball has someone for whom he should bring a present according to some permutation *p*, *pi* ≠ *i* for all *i*.

Unfortunately, Balls are quite clumsy. We know earlier that exactly *k* of them will forget to bring their gift. A Ball number *i* will get his present if the following two constraints will hold:

1. Ball number *i* will bring the present he should give.
2. Ball *x* such that *px* = *i* will bring his present.

What is minimum and maximum possible number of kids who will **not** get their present if exactly *k* Balls will forget theirs?

**Input**

The first line of input contains two integers *n* and *k* (2 ≤ *n* ≤ 106, 0 ≤ *k* ≤ *n*), representing the number of Balls and the number of Balls who will forget to bring their presents.

The second line contains the permutation *p* of integers from 1 to *n*, where *pi* is the index of Ball who should get a gift from the *i*-th Ball. For all *i*, *pi* ≠ *i* holds.

**Output**

You should output two values — minimum and maximum possible number of Balls who will **not** get their presents, in that order.

**Examples**

**input**

5 2  
3 4 1 5 2

**output**

2 4

**input**

10 1  
2 3 4 5 6 7 8 9 10 1

**output**

2 2

**Note**

In the first sample, if the third and the first balls will forget to bring their presents, they will be th only balls not getting a present. Thus the minimum answer is 2. However, if the first ans the second balls will forget to bring their presents, then only the fifth ball will get a present. So, the maximum answer is 4.

E. PolandBall and White-Red graph

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

PolandBall has an undirected simple graph consisting of *n* vertices. Unfortunately, it has no edges. The graph is very sad because of that. PolandBall wanted to make it happier, adding some red edges. Then, he will add white edges in every remaining place. Therefore, the final graph will be a clique in two colors: white and red.

Colorfulness of the graph is a value *min*(*dr*, *dw*), where *dr* is the diameter of the red subgraph and *dw* is the diameter of white subgraph. The diameter of a graph is a largest value *d* such that shortest path between some pair of vertices in it is equal to *d*. If the graph is not connected, we consider its diameter to be -1.

PolandBall wants the final graph to be as neat as possible. He wants the final colorfulness to be equal to *k*. Can you help him and find any graph which satisfies PolandBall's requests?

**Input**

The only one input line contains two integers *n* and *k* (2 ≤ *n* ≤ 1000, 1 ≤ *k* ≤ 1000), representing graph's size and sought colorfulness.

**Output**

If it's impossible to find a suitable graph, print -1.

Otherwise, you can output any graph which fulfills PolandBall's requirements. First, output *m* — the number of red edges in your graph. Then, you should output *m* lines, each containing two integers *ai* and *bi*, (1 ≤ *ai*, *bi* ≤ *n*, *ai* ≠ *bi*) which means that there is an undirected red edge between vertices *ai* and *bi*. Every red edge should be printed exactly once, you can print the edges and the vertices of every edge in arbitrary order.

Remember that PolandBall's graph should remain simple, so no loops or multiple edges are allowed.

**Examples**

**input**

4 1

**output**

-1

**input**

5 2

**output**

4  
1 2  
2 3  
3 4  
4 5

**Note**

In the first sample case, no graph can fulfill PolandBall's requirements.

In the second sample case, red graph is a path from 1 to 5. Its diameter is 4. However, white graph has diameter 2, because it consists of edges 1-3, 1-4, 1-5, 2-4, 2-5, 3-5.

D. PolandBall and Polygon

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

PolandBall has such a convex polygon with *n* veritces that no three of its diagonals intersect at the same point. PolandBall decided to improve it and draw some red segments.

He chose a number *k* such that *gcd*(*n*, *k*) = 1. Vertices of the polygon are numbered from 1 to *n* in a clockwise way. PolandBall repeats the following process *n* times, starting from the vertex 1:

Assume you've ended last operation in vertex *x* (consider *x* = 1 if it is the first operation). Draw a new segment from vertex *x* to *k*-th next vertex in clockwise direction. This is a vertex*x* + *k* or *x* + *k* - *n* depending on which of these is a valid index of polygon's vertex.

Your task is to calculate number of polygon's sections after each drawing. A section is a clear area inside the polygon bounded with drawn diagonals or the polygon's sides.

**Input**

There are only two numbers in the input: *n* and *k* (5 ≤ *n* ≤ 106, 2 ≤ *k* ≤ *n* - 2, *gcd*(*n*, *k*) = 1).

**Output**

You should print *n* values separated by spaces. The *i*-th value should represent number of polygon's sections after drawing first *i* lines.

**Examples**

**input**

5 2

**output**

2 3 5 8 11

**input**

10 3

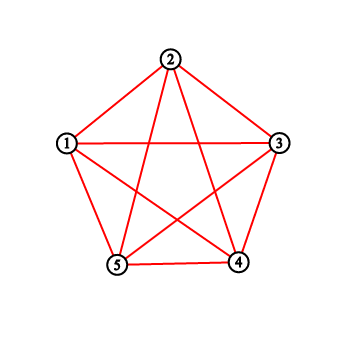
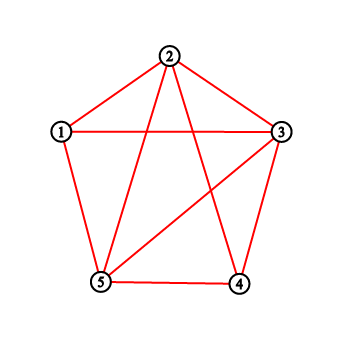
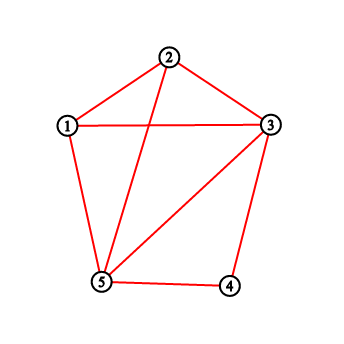
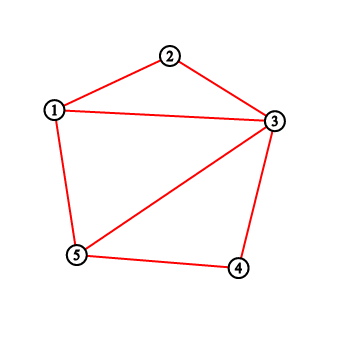
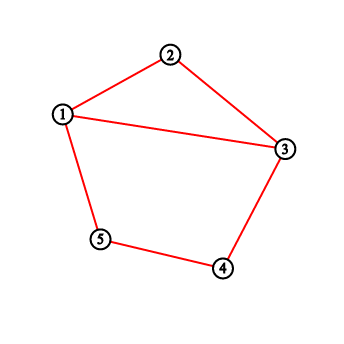
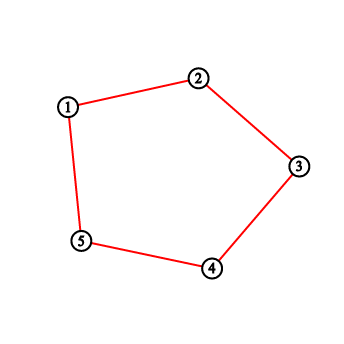
**output**

2 3 4 6 9 12 16 21 26 31

**Note**

The greatest common divisor (gcd) of two integers *a* and *b* is the largest positive integer that divides both *a* and *b* without a remainder.

For the first sample testcase, you should output "2 3 5 8 11". Pictures below correspond to situations after drawing lines.



C. PolandBall and Forest

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

PolandBall lives in a forest with his family. There are some trees in the forest. Trees are undirected acyclic graphs with *k* vertices and *k* - 1 edges, where *k* is some integer. Note that one vertex **is** a valid tree.

There is exactly one relative living in each vertex of each tree, they have unique ids from 1 to *n*. For each Ball *i* we know the id of its most distant relative living on the same tree. If there are several such vertices, we only know the value of the one with smallest id among those.

How many trees are there in the forest?

**Input**

The first line contains single integer *n* (1 ≤ *n* ≤ 104) — the number of Balls living in the forest.

The second line contains a sequence *p*1, *p*2, ..., *pn* of length *n*, where (1 ≤ *pi* ≤ *n*) holds and *pi* denotes the most distant from Ball *i*relative living on the same tree. If there are several most distant relatives living on the same tree, *pi* is the id of one with the smallest id.

It's guaranteed that the sequence *p* corresponds to some valid forest.

**Hacking**: To hack someone, you should provide a **correct forest** as a test. The sequence *p* will be calculated according to the forest and given to the solution you try to hack as input. Use the following format:

In the first line, output the integer *n* (1 ≤ *n* ≤ 104) — the number of Balls and the integer *m* (0 ≤ *m* < *n*) — the total number of edges in the forest. Then *m* lines should follow. The *i*-th of them should contain two integers *ai* and *bi* and represent an edge between vertices in which relatives *ai* and *bi* live. For example, the first sample is written as follows:

5 3  
1 2  
3 4  
4 5

**Output**

You should output the number of trees in the forest where PolandBall lives.

**Interaction**

From the technical side, this problem is interactive. However, it should not affect you (except hacking) since there is no interaction.

**Examples**

**input**

5  
2 1 5 3 3

**output**

2

**input**

1  
1

**output**

1

**Note**

In the first sample testcase, possible forest is: 1-2 3-4-5.

There are 2 trees overall.

In the second sample testcase, the only possible graph is one vertex and no edges. Therefore, there is only one tree.

B. PolandBall and Game

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

PolandBall is playing a game with EnemyBall. The rules are simple. Players have to say words in turns. You cannot say a word which was already said. PolandBall starts. The Ball which can't say a new word loses.

You're given two lists of words familiar to PolandBall and EnemyBall. Can you determine who wins the game, if both play optimally?

**Input**

The first input line contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 103) — number of words PolandBall and EnemyBall know, respectively.

Then *n* strings follow, one per line — words familiar to PolandBall.

Then *m* strings follow, one per line — words familiar to EnemyBall.

Note that one Ball **cannot** know a word more than once (strings are unique), but some words **can** be known by both players.

Each word is non-empty and consists of no more than 500 lowercase English alphabet letters.

**Output**

In a single line of print the answer — "YES" if PolandBall wins and "NO" otherwise. Both Balls play optimally.

**Examples**

**input**

5 1  
polandball  
is  
a  
cool  
character  
nope

**output**

YES

**input**

2 2  
kremowka  
wadowicka  
kremowka  
wiedenska

**output**

YES

**input**

1 2  
a  
a  
b

**output**

NO

**Note**

In the first example PolandBall knows much more words and wins effortlessly.

In the second example if PolandBall says kremowka first, then EnemyBall cannot use that word anymore. EnemyBall can only say wiedenska. PolandBall says wadowicka and wins.

A. PolandBall and Hypothesis

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

PolandBall is a young, clever Ball. He is interested in prime numbers. He has stated a following hypothesis: "There exists such a positive integer *n* that for each positive integer *m* number *n*·*m* + 1 is a prime number".

Unfortunately, PolandBall is not experienced yet and doesn't know that his hypothesis is incorrect. Could you prove it wrong? Write a program that finds a counterexample for any *n*.

**Input**

The only number in the input is *n* (1 ≤ *n* ≤ 1000) — number from the PolandBall's hypothesis.

**Output**

Output such *m* that *n*·*m* + 1 is not a prime number. Your answer will be considered correct if you output any suitable *m* such that 1 ≤ *m* ≤ 103. It is guaranteed the the answer exists.

**Examples**

**input**

3

**output**

1

**input**

4

**output**

2

**Note**

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself.

For the first sample testcase, 3·1 + 1 = 4. We can output 1.

In the second sample testcase, 4·1 + 1 = 5. We cannot output 1 because 5 is prime. However, *m* = 2 is okay since 4·2 + 1 = 9, which is not a prime number.

E. Dasha and cyclic table

time limit per test

6 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Dasha is fond of challenging puzzles: Rubik's Cube 3 × 3 × 3, 4 × 4 × 4, 5 × 5 × 5 and so on. This time she has a cyclic table of size *n* × *m*, and each cell of the table contains a lowercase English letter. Each cell has coordinates (*i*, *j*) (0 ≤ *i* < *n*, 0 ≤ *j* < *m*). The table is cyclic means that to the right of cell (*i*, *j*) there is the cell http://codeforces.com/predownloaded/8d/c7/8dc770e3f2a92bff2c4de0afd20bed6dac022ea4.png, and to the down there is the cell http://codeforces.com/predownloaded/94/20/94202fe965b30f7b1ee387b9655cbaae9fc0aaf0.png.

Dasha has a pattern as well. A pattern is a non-cyclic table of size *r* × *c*. Each cell is either a lowercase English letter or a question mark. Each cell has coordinates (*i*, *j*) (0 ≤ *i* < *r*, 0 ≤ *j* < *c*).

The goal of the puzzle is to find all the appearance positions of the pattern in the cyclic table.

We say that the cell (*i*, *j*) of cyclic table is an appearance position, if for every pair (*x*, *y*) such that 0 ≤ *x* < *r* and 0 ≤ *y* < *c* one of the following conditions holds:

* There is a question mark in the cell (*x*, *y*) of the pattern, or
* The cell http://codeforces.com/predownloaded/b6/9d/b69db58d1f79578693992aa725a888cd1b05cca6.png of the cyclic table equals to the cell (*x*, *y*) of the pattern.

Dasha solved this puzzle in no time, as well as all the others she ever tried. Can you solve it?.

**Input**

The first line contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 400) — the cyclic table sizes.

Each of the next *n* lines contains a string of *m* lowercase English characters — the description of the cyclic table.

The next line contains two integers *r* and *c* (1 ≤ *r*, *c* ≤ 400) — the sizes of the pattern.

Each of the next *r* lines contains a string of *c* lowercase English letter and/or characters '?' — the description of the pattern.

**Output**

Print *n* lines. Each of the *n* lines should contain *m* characters. Each of the characters should equal '0' or '1'.

The *j*-th character of the *i*-th (0-indexed) line should be equal to '1', in case the cell (*i*, *j*) is an appearance position, otherwise it should be equal to '0'.

**Examples**

**input**

5 7  
qcezchs  
hhedywq  
wikywqy  
qckrqzt  
bqexcxz  
3 2  
??  
yw  
?q

**output**

0000100  
0001001  
0000000  
0000000  
0000000

**input**

10 10  
fwtoayylhw  
yyaryyjawr  
ywrdzwhscy  
hnsyyxiphn  
bnjwzyyjvo  
kkjgseenwn  
gvmiflpcsy  
lxvkwrobwu  
wyybbcocyy  
yysijsvqry  
2 2  
??  
yy

**output**

1000100000  
0000000001  
0001000000  
0000010000  
0000000000  
0000000000  
0000000000  
0100000010  
1000000001  
0000010000

**input**

8 6  
ibrgxl  
xqdcsg  
okbcgi  
tvpetc  
xgxxig  
igghzo  
lmlaza  
gpswzv  
1 4  
gx??

**output**

000100  
000001  
000000  
000000  
010001  
000000  
000000  
000000D. Fedor and coupons

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

All our characters have hobbies. The same is true for Fedor. He enjoys shopping in the neighboring supermarket.

The goods in the supermarket have unique integer ids. Also, for every integer there is a product with id equal to this integer. Fedor has *n*discount coupons, the *i*-th of them can be used with products with ids ranging from *li* to *ri*, inclusive. Today Fedor wants to take exactly *k* coupons with him.

Fedor wants to choose the *k* coupons in such a way that the number of such products *x* that all coupons can be used with this product *x*is as large as possible (for better understanding, see examples). Fedor wants to save his time as well, so he asks you to choose coupons for him. Help Fedor!

**Input**

The first line contains two integers *n* and *k* (1 ≤ *k* ≤ *n* ≤ 3·105) — the number of coupons Fedor has, and the number of coupons he wants to choose.

Each of the next *n* lines contains two integers *li* and *ri* ( - 109 ≤ *li* ≤ *ri* ≤ 109) — the description of the *i*-th coupon. The coupons can be equal.

**Output**

In the first line print single integer — the maximum number of products with which all the chosen coupons can be used. The products with which at least one coupon cannot be used shouldn't be counted.

In the second line print *k* distinct integers *p*1, *p*2, ..., *pk* (1 ≤ *pi* ≤ *n*) — the ids of the coupons which Fedor should choose.

If there are multiple answers, print any of them.

**Examples**

**input**

4 2  
1 100  
40 70  
120 130  
125 180

**output**

31  
1 2

**input**

3 2  
1 12  
15 20  
25 30

**output**

0  
1 2

**input**

5 2  
1 10  
5 15  
14 50  
30 70  
99 100

**output**

21  
3 4

**Note**

In the first example if we take the first two coupons then all the products with ids in range [40, 70] can be bought with both coupons. There are 31 products in total.

In the second example, no product can be bought with two coupons, that is why the answer is 0. Fedor can choose any two coupons in this example.

C. Vladik and chat

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Recently Vladik discovered a new entertainment — coding bots for social networks. He would like to use machine learning in his bots so now he want to prepare some learning data for them.

At first, he need to download *t* chats. Vladik coded a script which should have downloaded the chats, however, something went wrong. In particular, some of the messages have no information of their sender. It is known that if a person sends several messages in a row, they all are merged into a single message. It means that **there could not be two or more messages in a row with the same sender**. Moreover, **a sender never mention himself in his messages**.

Vladik wants to recover senders of all the messages so that each two neighboring messages will have different senders and no sender will mention himself in his messages.

He has no idea of how to do this, and asks you for help. Help Vladik to recover senders in each of the chats!

**Input**

The first line contains single integer *t* (1 ≤ *t* ≤ 10) — the number of chats. The *t* chats follow. Each chat is given in the following format.

The first line of each chat description contains single integer *n* (1 ≤ *n* ≤ 100) — the number of users in the chat.

The next line contains *n* space-separated distinct usernames. Each username consists of lowercase and uppercase English letters and digits. The usernames can't start with a digit. Two usernames are different even if they differ only with letters' case. The length of username is positive and doesn't exceed 10 characters.

The next line contains single integer *m* (1 ≤ *m* ≤ 100) — the number of messages in the chat. The next *m* line contain the messages in the following formats, one per line:

* <username>:<text> — the format of a message with known sender. The username should appear in the list of usernames of the chat.
* <?>:<text> — the format of a message with unknown sender.

The text of a message can consist of lowercase and uppercase English letter, digits, characters '.' (dot), ',' (comma), '!'(exclamation mark), '?' (question mark) and ' ' (space). The text doesn't contain trailing spaces. The length of the text is positive and doesn't exceed 100 characters.

We say that a text mention a user if his username appears in the text as a word. In other words, the username appears in a such a position that the two characters before and after its appearance either do not exist or are not English letters or digits. For example, the text "Vasya, masha13 and Kate!" can mention users "Vasya", "masha13", "and" and "Kate", but not "masha".

It is guaranteed that in each chat **no known sender mention himself in his messages** and there are no two neighboring messages with the same known sender.

**Output**

Print the information about the *t* chats in the following format:

If it is not possible to recover senders, print single line "Impossible" for this chat. Otherwise print *m* messages in the following format:

<username>:<text>

If there are multiple answers, print any of them.

**Examples**

**input**

1  
2  
Vladik netman  
2  
?: Hello, Vladik!  
?: Hi

**output**

netman: Hello, Vladik!  
Vladik: Hi

**input**

1  
2  
netman vladik  
3  
netman:how are you?  
?:wrong message  
vladik:im fine

**output**

Impossible

**input**

2  
3  
netman vladik Fedosik  
2  
?: users are netman, vladik, Fedosik  
vladik: something wrong with this chat  
4  
netman tigerrrrr banany2001 klinchuh  
4  
?: tigerrrrr, banany2001, klinchuh, my favourite team ever, are you ready?  
klinchuh: yes, coach!  
?: yes, netman  
banany2001: yes of course.

**output**

Impossible  
netman: tigerrrrr, banany2001, klinchuh, my favourite team ever, are you ready?  
klinchuh: yes, coach!  
tigerrrrr: yes, netman  
banany2001: yes of course.

B. Ilya and tic-tac-toe game

754B

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Ilya is an experienced player in tic-tac-toe on the 4 × 4 field. He always starts and plays with Xs. He played a lot of games today with his friend Arseny. The friends became tired and didn't finish the last game. It was Ilya's turn in the game when they left it. Determine whether Ilya could have won the game by making single turn or not.

The rules of tic-tac-toe on the 4 × 4 field are as follows. Before the first turn all the field cells are empty. The two players take turns placing their signs into empty cells (the first player places Xs, the second player places Os). The player who places Xs goes first, the another one goes second. The winner is the player who first gets **three of his signs in a row next to each other** (horizontal, vertical or diagonal).

**Input**

The tic-tac-toe position is given in four lines.

Each of these lines contains four characters. Each character is '.' (empty cell), 'x' (lowercase English letter *x*), or 'o' (lowercase English letter *o*). It is guaranteed that the position is reachable playing tic-tac-toe, and it is Ilya's turn now (in particular, it means that the game is not finished). It is possible that all the cells are empty, it means that the friends left without making single turn.

**Output**

Print single line: "YES" in case Ilya could have won by making single turn, and "NO" otherwise.

**Examples**

**input**

xx..  
.oo.  
x...  
oox.

**output**

YES

**input**

x.ox  
ox..  
x.o.  
oo.x

**output**

NO

**input**

x..x  
..oo  
o...  
x.xo

**output**

YES

**input**

o.x.  
o...  
.x..  
ooxx

**output**

NO

**Note**

In the first example Ilya had two winning moves: to the empty cell in the left column and to the leftmost empty cell in the first row.

In the second example it wasn't possible to win by making single turn.

In the third example Ilya could have won by placing X in the last row between two existing Xs.

In the fourth example it wasn't possible to win by making single turn.