Problem A. CosmoCraft

Input file:	`cosmo.in`
Output file:	`cosmo.out`
Time limit:	3 seconds
Memory limit:	256 mebibytes

In the two-player game CosmoCraft you manage an economy in the hopes of producing an army capable of defeating your opponent. You manage the construction of workers, production facilities, and army units; the game revolves around balancing the resources you allocate to each. The game progresses in turns.

- Workers give you income at the rate of 1 dollar per turn.
- Production facilities let you produce either an army unit or a worker for the cost of 1 dollar. (only 1 army unit or worker can be produced per turn per facility)
- It costs 1 dollar to create a production facility.
- Your army, of course, lets you fight against your opponent.

You start off with n workers and k production facilities. The game progresses in turns — at each turn, you can spend the income you get from your workers on a mixture of workers, army, and creating production facilities. Workers produced this round do not give you income until the next round; likewise, production facilities do not become active until the next round. Any unspent income from the current round carries over to the next.

At the end of a round, you can take the total army you've produced and attack your opponent; if you have strictly more units than your opponent, the opponent loses immediately, and you retain the difference of the army sizes. Otherwise, your army is crushed and your opponent is left with the difference of the army sizes. (it would be wise for him to counter-attack after this, but you don't lose immediately at least). The game ends after t turns, at which point both players will usually attack with the larger army reigning victorious.

You're playing against your friend, and since you've played against him so many times you know exactly what he's going to spend his money on at every turn, and exactly when he's going to attack. Knowing this, you've decided that the best strategy is to play defensively — you just want to survive every attack, and amass as large an army in the meantime so you can counterattack (and hopefully win) at the end of the game.

What's the largest army you can have at the end of the game, given that you must survive all your friend's attacks?

Input

There will be several (50 or less) test cases in the input. Each test case will begin with a line with three integers: n, k and t, where n ($1 \le n \le 100$) is the number of workers you start with, k ($1 \le k \le 100$) is the number of production facilities you have at the start, and t ($1 \le t \le 10,000$) is the number of turns. On the next line will be $t-1$ integers, a_i ($0 \le a_i \le 2^{63}-1$), separated by single spaces. The i-th integer indicates the strength of the attack (that is, the number of army units your opponent is using in that attack) on turn i. The input will end with a line with three zeroes.

Output

For each test case output a single integer indicating the maximum number of armies you could have at the end of the game. Output -1 if it is impossible to survive. Output each integer on its own line, with no

spaces, and do not print any blank lines between answers. While it is possible for some inputs to generate unreasonably large answers, all judge inputs yield answers which will fit in a signed 64-bit integer.

Example

cosmo.in	cosmo.out
8 4 6 22 6 10 14 0 4 3 3 0 0 6 9 7 0 0 11 0 7 0 0 0 0	-1 11 101

Problem B. Covered Walkway

Input file:	`covered.in`
Output file:	`covered.out`
Time limit:	30 seconds
Memory limit:	256 Mebibytes

Your university wants to build a new walkway, and they want at least part of it to be covered. There are certain points which must be covered. It doesn't matter if other points along the walkway are covered or not.

The building contractor has an interesting pricing scheme. To cover the walkway from a point at x to a point at y, they will charge $c + (x - y)^2$, where c is a constant. Note that it is possible for $x = y$. If so, then the contractor would simply charge c. Given the points along the walkway and the constant c, what is the minimum cost to cover the walkway?

Input

There will be several (15 or less) test cases in the input. Each test case will begin with a line with two integers, n ($1 \le n \le 10^6$) and c ($1 \le c \le 10^9$), where n is the number of points which must be covered, and c is the contractor's constant. Each of the following n lines will contain a single integer, representing a point along the walkway that must be covered. The points will be in order, from smallest to largest. All of the points will be in the range from 1 to 10^9, inclusive. The input will end with a line with two zeroes.

Output

For each test case, output a single integer, representing the minimum cost to cover all of the specified points. Output each integer on its own line, with no spaces, and do not print any blank lines between answers. All possible inputs yield answers which will fit in a signed 64-bit integer.

Example

covered.in	covered.out
10 5000 1 23 45 67 101 124 560 789 990 1019 0 0	30726

Problem C. Double Dealing

Input file:	doubledealing.in
Output file:	doubledealing.out
Time limit:	25 seconds
Memory limit:	256 mebibytes

Take a deck of n unique cards. Deal the entire deck out to k players in the usual way: the top card to player 1, the next to player 2, the k-th to player k, the $k+1$-st to player 1, and so on. Then pick up the cards — place player 1's cards on top, then player 2, and so on, so that player k's cards are on the bottom. Each player's cards are in reverse order — the last card that they were dealt is on the top, and the first on the bottom.

How many times, including the first, must this process be repeated before the deck is back in its original order?

Input

There will be multiple pairwise distinct test cases in the input. Each case will consist of a single line with two integers, n and k ($1 \le n \le 800,\ 1 \le k \le 800$). The input will end with a line with two zeroes.

Output

For each test case in the input, print a single integer, indicating the number of deals required to return the deck to its original order. Output each integer on its own line, with no extra spaces, and no blank lines between answers. All possible inputs yield answers which will fit in a signed 64-bit integer.

Example

doubledealing.in	doubledealing.out
1 3 10 3 52 4 0 0	1 4 13

Problem D. The End of the World

Input file:	endofworld.in
Output file:	endofworld.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Legend says that there is a group of monks who are solving a large Towers of Hanoi puzzle. The Towers of Hanoi is a well-known puzzle, consisting of three pegs, with a stack of disks, each a different size. At the start, all of the disks are stacked on one of the pegs, and ordered from largest (on the bottom) to smallest (on the top). The object is to move this stack of disks to another peg, subject to two rules: 1) you can only move one disk at a time, and 2) you cannot move a disk onto a peg if that peg already has a smaller disk on it.

The monks believe that when they finish, the world will end. Suppose you know how far they've gotten. Assuming that the monks are pursuing the most efficient solution, how much time does the world have left?

Input

There will be several (2500 or less) test cases in the input. Each test case will consist of a string of length 1 to 63, on a single line. This string will contain only (capital) `A`'s, `B`'s and `C`'s. The length of the string indicates the number of disks, and each character indicates the position of one disk. The first character tells the position of the smallest disk, the second character tells the position of the second smallest disk, and so on, until the last character, which tells the position of the largest disk. The character will be `A`, `B` or `C`, indicating which peg the disk is currently on. You may assume that the monks overall goal is to move the disks from peg `A` to peg `B`, and that the input represents a legitimate position in the optimal solution. The input will end with a line with a single capital `X`.

Output

For each test case, print a single number on its own line indicating the number of moves remaining until the given Towers of Hanoi problem is solved. Output no extra spaces, and do not separate answers with blank lines. All possible inputs yield answers which will fit in a signed 64-bit integer.

Example

endofworld.in	endofworld.out
AAA BBB X	7 0

Problem E. Estimation

Input file:	estimate.in
Output file:	estimate.out
Time limit:	16 seconds
Memory limit:	256 mebibytes

"There are too many numbers here!" your boss bellows. "How am I supposed to make sense of all of this? Pare it down! Estimate!"

You are disappointed. It took a lot of work to generate those numbers. But, you'll do what your boss asks.

You decide to estimate in the following way: You have an array A of numbers. You will partition it into k contiguous sections, which won't necessarily be of the same size. Then, you'll use a single number to estimate an entire section. In other words, for your array A of size n, you want to create another array B of size n, which has k contiguous sections. If i and j are in the same section, then $B[i] = B[j]$. You want to minimize the error, expressed as the sum of the absolute values of the differences ($\sum |A[i] - B[i]|$).

Input

There will be several (25 or less) test cases in the input. Each test case will begin with two integers on a line, n ($1 \le n \le 2,000$) and k ($1 \le k \le 25,\ k \le n$), where n is the size of the array, and k is the number of contiguous sections to use in estimation. The array A will be on the next n lines, one integer per line. Each integer element of A will be in the range from $-10,000$ to $10,000$, inclusive. The input will end with a line with two zeroes.

Output

For each test case, output a single integer on its own line, which is the minimum error you can achieve. Output no extra spaces, and do not separate answers with blank lines. All possible inputs yield answers which will fit in a signed 64-bit integer.

Example

estimate.in	estimate.out
7 2 6 5 4 3 2 1 7 0 0	9

Problem F. Juggler

Input file:	juggler.in
Output file:	juggler.out
Time limit:	4 seconds
Memory limit:	256 mebibytes

As part of my magical juggling act, I am currently juggling a number of objects in a circular path with one hand. However, as my rather elaborate act ends, I wish to drop all of the objects in a specific order, in a minimal amount of time. On each move, I can either rotate all of the objects counterclockwise by one, clockwise by one, or drop the object currently in my hand. If I drop the object currently in my hand, the next object (clockwise) will fall into my hand. What's the minimum number of moves it takes to drop all of the balls I'm juggling?

Input

There will be several (30 or less) test cases in the input. Each test case begins with an integer n, ($1 \le n \le 10^5$) on its own line, indicating the total number of balls begin juggled. Each of the next n lines consists of a single integer, k_i ($1 \le k_i \le n$), which describes a single ball: i is the position of the ball starting clockwise from the juggler's hand, and k_i is the order in which the ball should be dropped. The set of numbers k_1, k_2, ..., k_n is guaranteed to be a permutation of the numbers $1 \ldots n$. The input will terminate with a line containing a single 0.

Output

For each test case, output a single integer on its own line, indicating the minimum number of moves I need to drop all of the balls in the desired order. Output no extra spaces, and do not separate answers with blank lines. All possible inputs yield answers which will fit in a signed 64-bit integer.

Example

juggler.in	juggler.out
3 3 2 1 0	5

Note

Explanation of the sample input: The first ball is in the juggler's hand and should be dropped third; the second ball is immediately clockwise from the first ball and should be dropped second; the third ball is immediately clockwise from the second ball and should be dropped first.

Problem G. Red-Blue Spanning Tree

Input file:	`redblue.in`
Output file:	`redblue.out`
Time limit:	10 seconds
Memory limit:	256 mebibytes

Given an undirected, unweighted, connected graph, where each edge is colored either blue or red, determine whether a spanning tree with exactly k blue edges exists.

Input

There will be several (40 or less) test cases in the input. Each test case will begin with a line with three integers: n, m and k, where n ($2 \le n \le 1000$) is the number of nodes in the graph, m (limited by structure of the graph) is the number of edges in the graph, and k ($0 \le k \le n$) is the number of blue edges desired in the spanning tree. Each of the next m lines will contain three elements, describing the edges: c, f ,t, where c is a character, either capital '`R`' or capital '`B`', indicating the color of the edge, and f and t are integers ($1 \le f, t \le n, t \ne f$) indicating the nodes that edge goes from and to. The graph is guaranteed to be connected, and there is guaranteed to be at most one edge between any pair of nodes. with a line with three zeroes.

Output

For each test case, output single line, containing 1 if it is possible to build a spanning tree with exactly k blue edges and 0 if it is not possible.

Example

redblue.in	redblue.out
3 3 2 B 1 2 B 2 3 R 3 1 2 1 1 R 1 2 0 0 0	1 0

Problem H. The Red Gem (Division 1 Only!)

Input file:	`redgem.in`
Output file:	`redgem.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

In circle land, in the museum of circles, a grand red circular gem is on display. The curator has decided to spice up the display, and has placed the gem on a purple circular platform, along with mundane orange circular gems. Starved citizens of circle land (points) have flocked to see the grand exhibit of the exquisite red gem. They cannot step on the purple exhibit floor, but can only stand on the circumference. Unfortunately, the mundane orange gems block the view of the exquisite red gem. Please help the museum folks determine the proportion of the circumference of the purple platform from which all of the red gem is visible, completely unobstructed by the orange gems.

Input

There will be several (1000 or less) test cases in the input. Each test case will begin with a line with five integers: n, p, x, y, r.

Where n ($1 \le n \le 100$) is the number of orange circles, p ($10 \le p \le 1,000$) is the radius of the purple platform, (x, y) is the center of the red gem relative to the center of the purple platform ($-1,000 \le x, y \le 1000$), and r ($0 \le r \le 1000$) is the radius of the red gem. The red gem is guaranteed to lie fully on the purple platform. No part of the red gem will extend past the purple platform. On each of the next n lines will be three integers x_i, y_i and r_i which represent the (x_i, y_i) center ($-1,000 \le x_i, y_i \le 1000$) relative to the center of the purple platform, and radius r_i ($0 < r_i \le 1000$) of each orange gem. As with the red gem, each orange gem is guaranteed to lie entirely on the purple platform. The orange gems will not overlap the red gem, and they will not overlap each other. The input will end with a line with five zeroes.

Output

For each test case, output a single floating point number on its own line, indicating the proportion of the perimeter of the purple platform where all of the red gem is visible. This result should be between 0 and 1 (inclusive). Output this number with 4 decimal places of accuracy. Output each number on its own line, with no spaces, and do not print any blank lines between answers.

Example

redgem.in	redgem.out
4 10 0 0 1 5 0 2 0 5 2 -5 0 2 0 -5 2 0 0 0 0 0	0.3082

Problem I. Science!

Input file:	`science.in`
Output file:	`science.out`
Time limit:	5 seconds
Memory limit:	256 Mebibytes

Welcome, ladies and gentlemen, to Aperture Science. Astronauts, War Heroes, Olympians — you're here because we want the best, and you are it. That said, it's time to make some science.

Now, I want each of you to stand on one of these buttons. Well done, we're making great progress here. Now let's do it again. Oh, come on — don't stand on the same button! Move, people! No, no, that button's only for the Astronauts, you know who you are. What?! You say you can't do everything I ask? Ok let's start over. You there, the Programmer, figure out how many times we can do this. And make it quick, we have a lot more science to get through...

Input

There will be several (70 or less) test cases in the input. The first line of each case will contain n ($2 \le n \le 80$) giving the number of people (and the number of buttons) in the experiment. The next n lines will contain n characters each. If the j-th character of the i-th line is '`Y`' it indicates that the i-th person can stand on the j-th button (it is '`N`' otherwise). The last line of input will be a 0.

Output

For each test case, output k, the maximum number of times everyone can be standing on buttons such that nobody stands on the same button more than once (This might be 0). After that, output k lines. Each line should contain n integers separated by single spaces, where the i-th integer describes which person is standing on the i-th button. All of the lines should be valid and none of them should put the same person on the same button as a previous line of the same test case. Output no extra spaces, and do not separate answers with blank lines. Note that correct outputs might not be unique.

Example

science.in	science.out
3 YYY NYY YNY 2 YN YN 0	2 3 1 2 1 2 3 0

Problem J. The Worm in the Apple (Division 1 Only!)

Input file:	worm.in
Output file:	worm.out
Time limit:	13 seconds
Memory limit:	256 mebibytes

Willy the Worm was living happily in an apple — until some vile human picked the apple, and started to eat it! Now, Willy must escape!

Given a description of the apple (defined as a convex shape in 3D space), and a list of possible positions in the apple for Willy (defined as 3D points), determine the minimum distance Willy must travel to get to the surface of the apple from each point.

Input

Input file will begin with a line with a single integer n ($4 \le n \le 1,000$), which tells the number of points describing the apple. On the next n lines will be three integers x, y and z ($-10,000 \le x, y, z \le 10,000$), where each point (x, y, z) is either on the surface of, or within, the apple. The apple is the convex hull of these points. No four points will be coplanar. Following the description of the apple, there will be a line with a single integer q ($1 \le q \le 10^5$), which is the number of queries — that is, the number of points where Willy might be inside the apple. Each of the following q lines will contain three integers x, y and z ($-10,000 \le x, y, z \le 10,000$), representing a point (x, y, z) where Willy might be. All of Willy's points are guaranteed to be inside the apple.

Output

For each query, output a single floating point number, indicating the minimum distance Willy must travel to exit the apple. Output this number with 4 decimal places of accuracy. Output each number on its own line, with no spaces, and do not print any blank lines between answers.

Example

worm.in	worm.out
6 0 0 0 100 0 0 0 100 0 0 0 100 20 20 20 30 20 10 4 1 1 1 30 30 35 7 8 9 90 2 2	1.0 2.8868 7.0 2.00

Problem K. Combinations (Division 2 Only!)

Input file:	`combinations.in`
Output file:	`combinations.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

Probability and statistics have numerous examples where people want to analyze how to select m items from a set of n items. For example, consider the following 5 items: 0, 1, 2, 3, and 4. Suppose you were interested in selecting 3 items from this list. If we were to list all the ways to select 3 items from this set, we could list these 10 possibilities: 012, 013, 014, 023, 024, 034, 123, 124, 134, 234.

There are many ways to order the elements of this set, but for this problem we are interested in listing the combinations lexicographically. That is, if we view the above list as 3-letter "words", 012 comes before 013 and 134. Note that each 3-letter combination is itself written lexicographically, as is the list of all 3-item combinations. That is, the digits of the item 014 are written in increasing order (not 041, for example).

The problem you are asked to solve is, given that you wish to consider all the ways to select M elements from a set of N elements, identify the i-th possibility, if all of those possibilities were listed lexicographically.

In this problem N can be as large as 36. So, we will use the uppercase letters to fill out the set of objects, where the ordering is: 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ. That is, the lexicographical ordering will use these 36 characters in determining "alphabetical" order. ('0' comes before 'A', for example).

Input

Each line of the input will represent one test case and will contain 3 integers: N, M, and I. N represents the total number of objects to select from, M represent the number of objects to select, and I represents the position of the ith combination, where the combinations are ordered as described above. N will be at least 1 and at most 36. M will be at least 1 and no more than N. I will be a legal value for M and N (i.e., at least 1 and at most $N!/(M!(N-M)!)$. The final line of input will be when N is equal to 0. Do not produce any output for this last line of input. There will be at most 2000 problems to solve in the input file.

Output

For each input, display the ith combination using the format shown below, with the problem number, followed by a colon and space, followed by the i-th combination.

Example

combinations.in	combinations.out
5 3 1 5 3 2 5 3 10 4 2 1 4 2 6 10 5 1 10 5 252 11 11 1 36 2 1 36 2 2 36 2 630 36 36 1 36 18 1 36 18 9075135300 0 0 0	1: 012 2: 013 3: 234 4: 01 5: 23 6: 01234 7: 56789 8: 0123456789A 9: 01 10: 02 11: YZ 12: 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ 13: 0123456789ABCDEFGH 14: IJKLMNOPQRSTUVWXYZ

Problem L. Find a cycle (Division 2 Only!)

Input file:	cycle.in
Output file:	cycle.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Consider the following number operation (let's call it the root-rounding operation) that begins with some positive integer, and ends with a positive integer:

1. Begin with a positive integer, n.

2. Compute the largest integer that is less than or equal to the square root of n. Call this result m.

3. If $n + m$ is even, then the final result is $n + m$. Otherwise, the final result is $n - m$.

One might consider taking a number, computing its root-rounded result, and then taking that result and computing its root-rounded result, and so on. For example, begin with 25, and the result is 30. Then, take 30 and compute its result: 25. 25 gives 30. 30 gives 25, and so on. This is referred to as a "2-cycle". On the other hand, if you begin with 24, the resulting sequence is a 13-cycle: 24, 28, 23, 19, 15, 18, 22, 26, 21, 17, 13, 16, 20, 24. Notice, that the last number creates a cycle. We refer to this particular cycle as a 13-cycle, because it contains exactly 13 non-repeating integers, with the 14th integer simply repeating the first integer.

Input

First line of input file contains integer T ($1 \le T \le 200$) — number of test cases. Each test case is given on separate line and contains one positive integer, with value at most 10^6. You will determine the length of each integer's root-rounding cycle.

Output

Formatting your results as shown in the sample output below (the input, followed by a colon and space, followed by the length of its root-rounding cycle).

Example

cycle.in	cycle.out
3 25 24 1	25: 2-cycle 24: 13-cycle 1: 2-cycle