2013 ACM ICPC
Southeast USA Regional
Programming Contest

2 November, 2013

Division 1

Hosted by:

Florida Institute of Technology

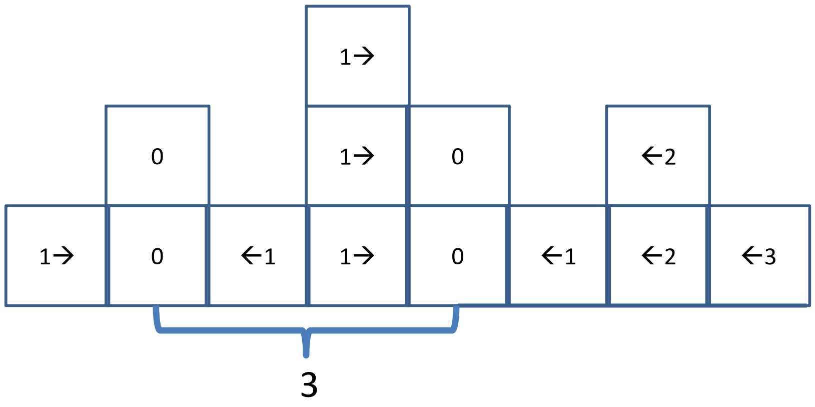
Georgia Institute of Technology

University of West Florida

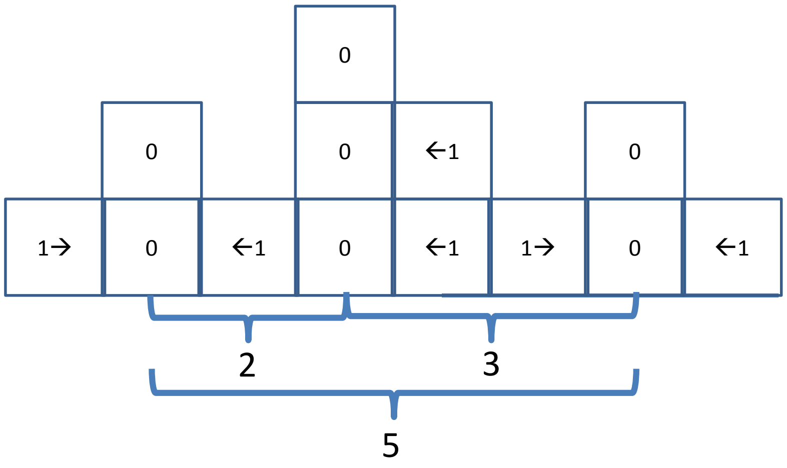
A: Beautiful Mountains

Paco loves playing with stacks of blocks. He likes to pretend that they are mountains, and he loves making his own terrain. Lately, he's been restricting himself to a particular way of rearranging the blocks. He puts all of his stacks of blocks into a straight line. Then, he only changes the arrangement one block at a time. Paco does this by finding two adjacent stacks of blocks and moving one block from one stack to the other.

Paco has made all sorts of arrangements of his 'mountains' using this technique. Now he has decided to make his most beautiful arrangements yet. Paco finds a mountain range beautiful if, for every pair of mountains, the distance between the two mountains is a prime number (that's every pair, not just every adjacent pair). A mountain range with a single stack is beautiful by default. Paco considers a stack of blocks to be a mountain if it has at least one block.



This diagram shows an initial configuration of the blocks, and a way to make two stacks, at a distance of three apart, with 13 moves. However, with only 6 moves, Paco can make a beautiful arrangement with 3 stacks:



Given a current arrangement of blocks, what is the minimum amount of effort needed for Paco to make it beautiful?

Input

There will be several test cases in the input. Each test case will begin with a line with one integer ***n*** (1≤***n***≤30,000), which is the number of stacks of blocks. On the next line will be ***n*** integers ***b*** (0≤***b***≤1,000), indicating the number of blocks in that stack. The integers will be separated by a single space, with no leading or trailing spaces. Note that every stack will be listed, even those with zero blocks. End of input will be marked by a line with a single **0**.

Output

For each test case, output a single integer indicating the least number of moves required to make Paco's stacks of blocks 'beautiful'. Output no spaces, and do not separate answers with blank lines.

Sample Input	**Sample Output**

```
5
1 2 1 2 1
8
1 2 1 3 2 1 2 1
0
```
	```
	3
	6
	```

B: Nested Palindromes

Palindromes are numbers that read the same forwards and backwards. Your friend Percy recently became interested in a special kind of palindrome that he calls a *Nested Palindrome*. A *Nested Palindrome* meets three conditions:

- The number is a palindrome.
- Split the number in the middle. The first half of the digits of the number is also a *Nested Palindrome*. If the number has an odd number of digits, don't consider the middle digit as part of the first half.
- No two adjacent digits are the same.

Percy says that he has written a *Nested Palindrome* with no leading zeros on a slip of paper. Next, Percy says that he has erased some of the digits in the number and replaced those digits with question marks. He asks you to think about all possible numbers, in increasing order, that can fill those digits and could possibly form the number Percy wrote. Of course, Percy may not be telling the truth about having written a *Nested Palindrome* in the first place.

Percy tells you that the number he wrote is the ***k***th number of this potentially large list. Your task is to find that ***k***th number.

Input

There will be several test cases in the input. Each test case will consist of two lines. The first line will contain an integer ***k*** ($1 \le k \le 10^{18}$), which is the position in the ordered list you must find. The second line contains a string of length 1 to 10,000, consisting only of digits ('0' to '9') and question marks ('?'). Input is terminated by a line with a single **0**.

Output

For each test case, output the *Nested Palindrome* that Percy is looking for. If that number does not exist, or if the string cannot form a *Nested Palindrome*, output **-1**. Output no spaces, and do not separate answers with blank lines.

Sample Input	**Sample Output**

`1`
`1?1`
`1`
`?3?`
`1`
`?1?`
`55`
`???`
`55`
`1?1`
`3`
`0?0`
`0`
	`101`
	`131`
	`212`
	`707`
	`-1`
	`-1`

C: Ping!

Suppose you are tracking some satellites. Each satellite broadcasts a 'Ping' at a regular interval, and the intervals are unique (that is, no two satellites ping at the same interval). You need to know which satellites you can hear from your current position. The problem is that the pings cancel each other out. If an even number of satellites ping at a given time, you won't hear anything, and if an odd number ping at a given time, it sounds like a single ping. All of the satellites ping at time 0, and then each pings regularly at its unique interval.

Given a sequence of pings and non-pings, starting at time 0, which satellites can you determine that you can hear from where you are? The sequence you're given may, or may not, be long enough to include all of the satellites' ping intervals. There may be satellites that ping at time 0, but the sequence isn't long enough for you to hear their next ping. You don't have enough information to report about these satellites. Just report about the ones with an interval short enough to be in the sequence of pings.

Input

There will be several test cases in the input. Each test case will consist of a single string on its own line, with from 2 to 1,000 characters. The first character represents time 0, the next represents time 1, and so on. Each character will either be a 0 or a 1, indicating whether or not a ping can be heard at that time (0=No, 1=Yes). Each input is guaranteed to have at least one satellite that can be heard. The input will end with a line with a single **0**.

Output

For each test case, output a list of integers on a single line, indicating the intervals of the satellites that you know you can hear. Output the intervals in order from smallest to largest, with a single space between them. Output no extra spaces, and do not separate answers with blank lines.

Sample Input	Sample Output
01000101101000	1 2 3 6 8 10 11 13
1001000101001000	3 6 7 12 14 15
0	

D: Electric Car Rally

In an attempt to demonstrate the practicality of electric cars, ElecCarCo is sponsoring a cross-country road rally. There are ***n*** charging stations for the rally where cars may check in and charge their batteries

The rally may require multiple days of travel. Each car can travel four hours (240 minutes) between charges. A car must be plugged into a charger for two minutes for each minute of travel time. Cars start the rally at noon on the first day, fully charged. Cars are permitted remain at a station even after they are fully charged.

It is only possible to drive directly between select pairs of stations. Variations in traffic conditions, road conditions, availability of HOV lanes, etc., result in different travel times along each route depending upon the time of day at which travel along that route begins. All roads are two-way, and the prevailing conditions affect travel in both directions.

The winner is the first car to reach checkpoint ***n***-1, starting form checkpoint 0**.** Other than the starting and ending conditions, cars may pass through the stations in any order, and need not visit all stations to complete the course.

Write a program to determine the earliest time, expressed as the total number of minutes elapsed since the start of the rally, at which a car could reach the final checkpoint.

The Input

There will be several test cases in the input. Each test case starts with a line containing ***n*** (1≤***n***≤500), the number of stations, and ***m*** (1≤***m***≤1,000), the number of connecting road segments.

This is followed by ***m*** blocks, each block describing one road segment. A road segment block has the following structure:

Each block begins with a single line containing two integers, ***a*** and ***b*** (0≤***a***,***b***≤***n***-1, ***a***≠***b***). These numbers are the two checkpoints connected by that segment. The connections are undirected: a segment permitting travel from station ***a*** to station ***b*** will also allow travel from station ***b*** to station ***a***.

This is followed by from one to twenty 'travel lines' describing travel times. Each of the travel lines contains 3 numbers: ***Start***, ***Stop***, (0≤***Start***<***Stop***≤1,439), and ***Time*** (0<***Time***<1,000). ***Start*** and ***Stop*** are the time of day (expressed in minutes since midnight) described by this line, and ***Time*** is the travel time, in minutes, required to traverse this road segment if travel begins at any time in the range [***Start***..***Stop***], inclusive. The first travel line in a block will have a start time of 0 (midnight, or 00:00). The final travel line in a block will have a stop time of 1439 (i.e., 23:59, or 1 less than 24 hours times 60 minutes). Adjacent travel lines in the input will be arranged in order, and the start time of any line after the first is one higher than the stop time of the preceding line. The travel lines will cover all times from 00:00 to 23:59.

Input will end with a line with two **0**s. All test cases will describe a course that can be completed by the cars.

The Output

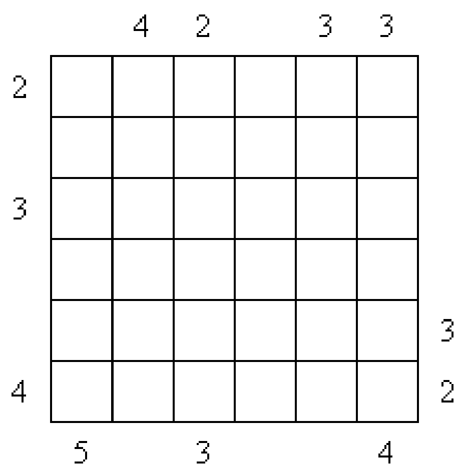
For each test case, output a single integer representing the smallest number of minutes needed to complete the rally. Output no spaces, and do not separate answers with blank lines.

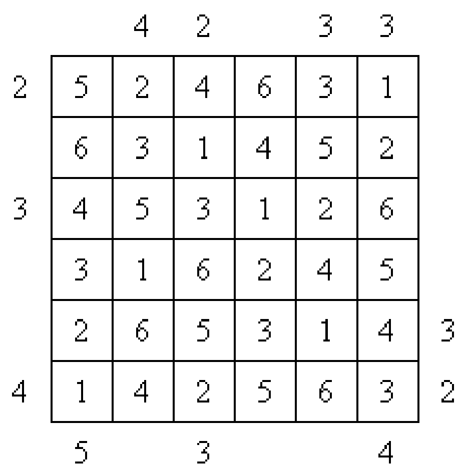
Sample Input	Sample Output

```
4 4
0 1
0 1439 100
0 2
0 1439 75
1 3
0 720 150
721 824 100
825 1000 75
1001 1439 150
2 3
0 1439 150
3 2
0 1
0 10 200
11 1439 300
1 2
0 10 200
11 1439 300
4 3
0 1
0 719 500
720 1439 240
1 2
0 964 500
965 1439 2
2 3
0 971 500
972 1439 3
0 0
```
	```
	180
	2360
	255
	```

E: Skyscrapers

Skyscrapers is a pencil puzzle. It's played on a square ***n*x*n*** grid. Each cell of the grid has a building. Each row, and each column, of the grid must have exactly one building of height 1, height 2, height 3, and so on, up to height ***n***. There may be numbers at the beginning and end of each row, and each column. They indicate how many buildings can be seen from that vantage point, where taller buildings obscure shorter buildings. In the game, you are given the numbers along the outside of the grid, and you must determine the heights of the buildings in each cell of the grid.





Consider a single row of a puzzle of size ***n*x*n***. If we know how many buildings can be seen from the left, and from the right, of the row, how many different ways are there of populating that row with buildings of heights 1..***n***?

Input

There will be several test cases in the input. Each test case consists of three integers n a single line: **n** (1≤***n***≤5,000), ***left*** (1≤***left***≤***n***), and ***right*** (1≤***right***≤***n***), where ***n*** is the size of the row, and ***left*** and ***right*** are the number of buildings that can be seen from the left and right, respectively. The Input will end with a line with three **0**s.

Output

For each test case, print a single integer indicating the number of permutations which satisfy the constraints, modulo 1,000,000,007 (that's not a misprint, the last digit is a seven). Output no extra spaces, and do not separate answers with blank lines.

Sample Input	**Sample Output**
3 2 2	2
4 1 2	2
0 0 0	

F: Star Simulations

In massive simulations of star systems, we don't want to have to model the gravitational effects of pairs of bodies that are too far away from each other, because that will take excess computing power (and their effects on each other are negligible). We want to only consider the effects two objects have on each other if the Euclidean distance between them is less than ***k***.

Given a list of ***n*** points in space, how many have a distance of less than ***k*** from each other?

Input

There will be several test cases in the input. Each test case will begin with a line with two integers, ***n*** ($2 \le n \le 100{,}000$) and ***k*** ($1 \le k \le 10^9$), where ***n*** is the number of points, and ***k*** is the desired maximum distance. On each of the following ***n*** lines will be three integers ***x***, ***y*** and ***z*** ($-10^9 \le x,y,z \le 10^9$) which are the (***x***,***y***,***z***) coordinates of one point. Within a test case, there will be no duplicate points. Since star systems are generally sparse, it is guaranteed that no more than 100,000 pairs of points will be within ***k*** of each other. The input will end with a line with two **0**s.

Output

For each test case, output a single integer indicating the number of unique pairs of points that are less than ***k*** apart from each other. Output no spaces, and do not separate answers with blank lines.

Sample Input	Sample Output

7 2
0 0 0
1 0 0
1 2 0
1 2 3
1000 1000 1000
1001 1001 1000
1001 999 1001
7 3
0 0 0
1 0 0
1 2 0
1 2 3
-1000 1000 -1000
-1001 1001 -1000
-1001 999 -1001
7 4
0 0 0
1 0 0
1 2 0
1 2 3
1000 -1000 1000
1001 -1001 1000
1001 -999 1001
0 0
	3
	6
	9

G: Tandem Repeats

Tandem Repeats occur in DNA when a pattern of one or more nucleotides is repeated, and the repetitions are directly adjacent to each other. For example, consider the sequence:

```
ATTCGATTCGATTCG
```

This contains 9 *Tandem Repeats*:

```
ATTCGATTCGATTCG
ATTCGATTCGATTCG
ATTCGATTCGATTCG
ATTCGATTCGATTCG
ATTCGATTCGATTCG
ATTCGATTCGATTCG
ATTCGATTCGATTCG
ATTCGATTCGATTCG
ATTCGATTCGATTCG
```

Given a nucleotide sequence, how many *Tandem Repeats* occur in it?

Input

There will be several test cases in the input. Each test case will consist of a single string on its own line, with from 1 to 100,000 capital letters, consisting only of **A**, **G**, **T** and **C**. This represents a nucleotide sequence. The input will end with a line with a single **0**.

Output

For each test case, output a single integer on its own line, indicating the number *Tandem Repeats* in the nucleotide sequence. Output no spaces, and do not separate answers with blank lines.

Sample Input	Sample Output
AGGA	1
AGAG	1
ATTCGATTCGATTCG	9
0	

H: Triangles

Given ***n*** points in a plane, find the triangles with the smallest and largest areas formed by any three of the points.

Input

There will be several test cases in the input. Each test case will begin with an integer ***n*** (3≤***n***≤2,000) on its own line, indicating the number of points. On each of the next ***n*** lines will be two integers ***x*** and ***y*** (-10,000≤***x***,***y***≤10,000), representing a point. No test case will contain duplicate points. The input will end with a **0** on its own line.

Output

For each case, output the areas of the smallest, then largest, triangles formed by any 3 of the points in the test case. Output these numbers with exactly one decimal place of accuracy, with exactly one space between them. Output no extra spaces, and do not separate answers with blank lines.

Sample Input

```
4
-5 -5
-4 3
4 1
3 -2
7
1 0
2 0
0 2
2 3
0 1
3 0
0 3
0
```

Sample Output

```
10.5 33.0
0.0 4.0
```

I: It Takes a Village

As a Sociologist, you are studying a certain kingdom. This kingdom has a capitol city, several villages, and roads between them all. Through your sociological studies, you have been able to determine that there are three separate conditions under which one village will economically affect another village. Village P will affect village Q if ANY of the following are true:

1. If there are two completely different paths to get from village P to village Q, with no villages in common (other than P and Q).
2. If every path from Q to the capitol goes through P.
3. If P affects village R and R affects Q.

The kingdom is starting to build trading posts, to boost the economic health of its villages. When it builds a trading post, it increases the overall revenue of the village it is placed in, and of all villages which are affected by that village according to the above rules. Now, the king wants to know the effect of his new trading posts, so he occasionally asks you to tell him the revenue of a certain village.

Given a sequence of the kings actions, both building trading posts and asking about a certain village, answer his questions.

Input

There will be several test cases in the input. Each test case will begin with a line with three integers, ***n*** (1≤***n***≤100,000), ***m*** (0≤***m***≤100,000), and ***q*** (1≤***q***≤200,000), where ***n*** is the number of villages, ***m*** is the number of roads, and ***q*** is the number of actions the king performs. The villages are numbered **1**..***n***, and **1** represents the capitol.

On each of the next ***m*** lines will be a two integers ***a***, ***b*** (1≤***a***,***b***≤***n***, ***a***!=***b***), representing a road from village ***a*** to village ***b***. The roads are two-way, supporting travel in either direction. It is possible to get from the capitol to every village by some route.

The next ***q*** lines represent the king's actions, in order, and will have one of two forms:

```
+ k x
```

Here, the king builds a trading post at village ***k*** (1≤***k***≤***n***), which increases all affected villages' revenues by ***x*** (1≤***x***≤1,000).

```
? k
```

Here, the king asks you for the total revenue for village ***k*** (1≤***k***≤***n***), including any trading posts in that village, and all villages that affect that village.

The parts of the king's commands will be separated by a single space, and will have no leading or trailing blanks. The input will end with a line with two **0**s.

Output

For each '? *k*' question the king asks, print a single integer in its own line, representing the answer to that question. Answer the king's questions in order. Output no spaces, and do not separate answers with blank lines.

Sample Input	Sample Output
3 2 4 1 2 3 1 + 1 1 ? 3 + 2 3 ? 3 2 2 4 1 2 2 1 + 1 1 ? 2 + 2 3 ? 1 0 0 0	1 1 1 4

J: You Win!

You just achieved the High Score on your favorite video game! Now, you get to enter your name! You have to use the controller to enter your name, which can be awkward. Here's how it works:

- There are only the 26 capital letters **A** to **Z**, in order. There are no numbers, spaces, lower case letters, or any other characters.
- Pushing UP or DOWN changes the active letter one letter forward (UP) or backward (DOWN). The active letter starts at **A**. It will not reset when you move around in the name. It also wraps: UP from **Z** goes to **A**, DOWN from **A** goes to **Z**.
- Pushing LEFT or RIGHT moves the cursor one letter left or right in the current name. Note that once the cursor is at either end of the current name, it cannot move any further in that direction.
- Pushing the FIRE button adds the active letter to the name.

For example, consider the name 'ALMA'. One way you could enter 'ALMA' is like this:

Action	# of Pushes	Name (| = Cursor)	Active Letter
FIRE	1	A|	A
UP	11	A|	L
FIRE	1	AL|	L
UP	1	AL|	M
FIRE	1	ALM|	M
DOWN	12	ALM|	A
FIRE	1	ALMA|	A

This would take 28 button pushes. However, consider entering 'ALMA' like this:

Action	# of Pushes	Name (| = Cursor)	Active Letter
FIRE	1	A|	A
FIRE	1	AA|	A
LEFT	1	A|A	A
UP	11	A|A	L
FIRE	1	AL|A	L
UP	1	AL|A	M
FIRE	1	ALM|A	M

This takes only 17 button pushes. Given a name, what is the fewest number of button pushes needed to enter that name? Assume that the active letter starts at **A**, and that it doesn't matter where the cursor ends up when you're done.

Input

There will be several test cases in the input. Each test case will consist of a single string on its own line, with from 1 to 18 capital letters, representing a name that must be entered into the High Score list. The input will end with a line with a single **0**.

Output

For each test case, output a single integer representing the smallest number of button pushes needed to enter the name. Output no spaces, and do not separate answers with blank lines.

Sample Input	Sample Output
ALMA	17
YES	21
0	