# Problem K: Through the Desert

Imagine you are an explorer trying to cross a desert. Many dangers and obstacles are waiting on your path. Your life depends on your trusty old jeep having a large enough fuel tank. But how large exactly does it have to be? Compute the smallest volume needed to reach your goal on the other side.

The following events describe your journey:

`Fuel consumption n`
: means that your truck needs *n* litres of gasoline per 100 km. *n* is an integer in the range [1..30]. Fuel consumption may change during your journey, for example when you are driving up or down a mountain.

`Leak`
: means that your truck's fuel tank was punctured by a sharp object. The tank will start leaking gasoline at a rate of 1 litre of fuel per kilometre. Multiple leaks add up and cause the truck to lose fuel at a faster rate.

However, not all events are troublesome in this desert. The following events increase your chances of survival:

`Gas station`
: lets you fill up your tank.

`Mechanic`
: fixes all the leaks in your tank.

And finally:

`Goal`
: means that you have safely reached the end of your journey!

## Input Format

The input consists of a series of test cases. Each test case consists of at most 50 events. Each event is described by an integer, the distance (in kilometres measured from the start) where the event happens, followed by the type of event as above.

In each test case, the first event is of the form '`0 Fuel consumption n`', and the last event is of the form '`d Goal`' (*d* is the distance to the goal).

Events are given in sorted order by non-decreasing distance from the start, and the '`Goal`' event is always the last one. There may be multiple events at the same distance—process them in the order given.

Input is terminated by a line containing '`0 Fuel consumption 0`' which should not be processed.

## Output Format

For each test case, print a line containing the smallest possible tank volume (in litres, with three digits precision after the decimal point) that will guarantee a successful journey.

## Sample Input

```
0 Fuel consumption 10
100 Goal
0 Fuel consumption 5
100 Fuel consumption 30
200 Goal
0 Fuel consumption 20
10 Leak
25 Leak
25 Fuel consumption 30
50 Gas station
70 Mechanic
100 Leak
120 Goal
0 Fuel consumption 0
```

## Sample Output

```
10.000
35.000
81.000
```

---

*Martin Müller*

# Problem C
## Ancient Messages
### Problem ID: ancient

In order to understand early civilizations, archaeologists often study texts written in ancient languages. One such language, used in Egypt more than 3000 years ago, is based on characters called hieroglyphs. Figure C.1 shows six hieroglyphs and their names. In this problem, you will write a program to recognize these six characters.

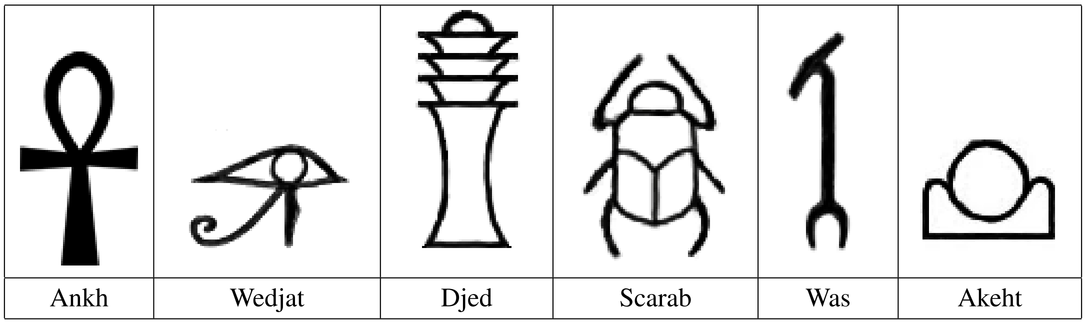


Figure C.1: Six hieroglyphs

## Input

The input consists of several test cases, each of which describes an image containing one or more hieroglyphs chosen from among those shown in Figure C.1. The image is given in the form of a series of horizontal scan lines consisting of black pixels (represented by 1) and white pixels (represented by 0). In the input data, each scan line is encoded in hexadecimal notation. For example, the sequence of eight pixels `10011100` (one black pixel, followed by two white pixels, and so on) would be represented in hexadecimal notation as `9c`. Only digits and lowercase letters `a` through `f` are used in the hexadecimal encoding. The first line of each test case contains two integers, $H$ and $W$. $H$ $(0 < H \leq 200)$ is the number of scan lines in the image. $W$ $(0 < W \leq 50)$ is the number of hexadecimal characters in each line. The next $H$ lines contain the hexadecimal characters of the image, working from top to bottom. Input images conform to the following rules:

- The image contains only hieroglyphs shown in Figure C.1.
- Each image contains at least one valid hieroglyph.
- Each black pixel in the image is part of a valid hieroglyph.
- Each hieroglyph consists of a connected set of black pixels and each black pixel has at least one other black pixel on its top, bottom, left, or right side.
- The hieroglyphs do not touch and no hieroglyph is inside another hieroglyph.
- Two black pixels that touch diagonally will always have a common touching black pixel.
- The hieroglyphs may be distorted but each has a shape that is topologically equivalent to one of the symbols in Figure C.1[1].

The last test case is followed by a line containing two zeros.

[1]Two figures are topologically equivalent if each can be transformed into the other by stretching without tearing.

## Output

For each test case, display its case number followed by a string containing one character for each hieroglyph recognized in the image, using the following code:

| | |
|---|---|
| Ankh: | `A` |
| Wedjat: | `J` |
| Djed: | `D` |
| Scarab: | `S` |
| Was: | `W` |
| Akhet: | `K` |

In each output string, print the codes in alphabetic order. Follow the format of the sample output.

The sample input contains descriptions of test cases shown in Figures C.2 and C.3. Due to space constraints not all of the sample input can be shown on this page.

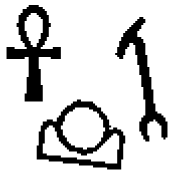


Figure C.2: AKW

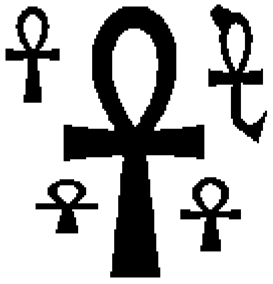
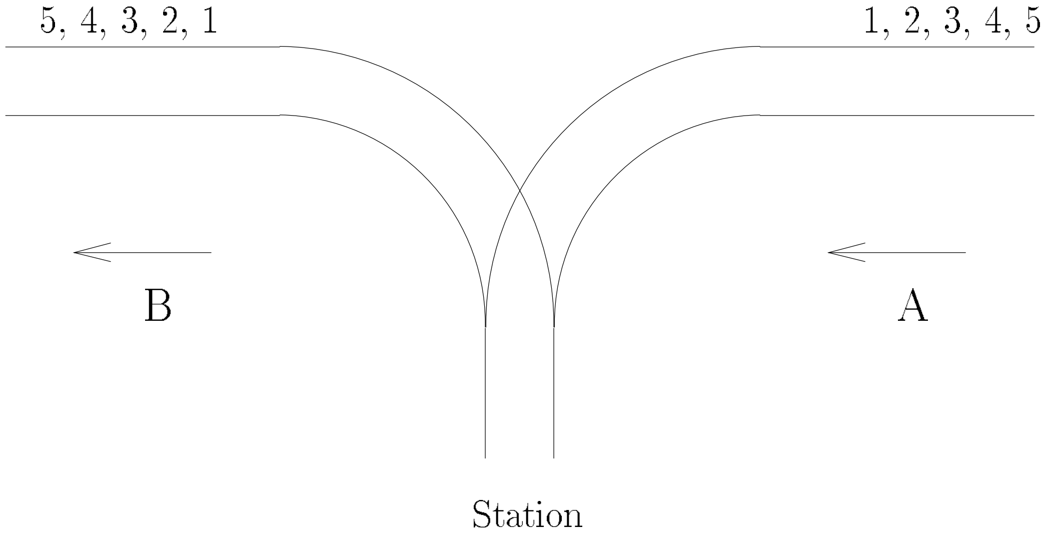


Figure C.3: AAAAA

| **Sample input** | **Output for the Sample Input** |
|---|---|
| `8 6`<br>`3c0800`<br>`241800`<br>`3c303c`<br>`181028`<br>`7e1066`<br>`18387e`<br>`182842`<br>`18007e`<br>`8 10`<br>`3c3c3c3c3c`<br>`2424242424`<br>`3c3c3c3c3c`<br>`1818181818`<br>`7e7e7e7e7e`<br>`1818181818`<br>`1818181818`<br>`1818181818` | `Case 1: AKW`<br>`Case 2: AAAAA` |

## 514 Rails

There is a famous railway station in PopPush City. Country there is incredibly hilly. The station was built in last century. Unfortunately, funds were extremely limited that time. It was possible to establish only a surface track. Moreover, it turned out that the station could be only a dead-end one (see picture) and due to lack of available space it could have only one track.



The local tradition is that every train arriving from the direction A continues in the direction B with coaches reorganized in some way. Assume that the train arriving from the direction A has $N \leq 1000$ coaches numbered in increasing order $1, 2, \ldots, N$. The chief for train reorganizations must know whether it is possible to marshal coaches continuing in the direction B so that their order will be $a_1.a_2, \ldots, a_N$. Help him and write a program that decides whether it is possible to get the required order of coaches. You can assume that single coaches can be disconnected from the train before they enter the station and that they can move themselves until they are on the track in the direction B. You can also suppose that at any time there can be located as many coaches as necessary in the station. But once a coach has entered the station it cannot return to the track in the direction A and also once it has left the station in the direction B it cannot return back to the station.

### Input

The input file consists of blocks of lines. Each block except the last describes one train and possibly more requirements for its reorganization. In the first line of the block there is the integer $N$ described above. In each of the next lines of the block there is a permutation of $1, 2, \ldots, N$ The last line of the block contains just 0.

The last block consists of just one line containing 0.

### Output

The output file contains the lines corresponding to the lines with permutations in the input file. A line of the output file contains `Yes` if it is possible to marshal the coaches in the order required on the corresponding line of the input file. Otherwise it contains `No`. In addition, there is one empty line after the lines corresponding to one block of the input file. There is no line in the output file corresponding to the last “null” block of the input file.

## Sample Input

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

## Sample Output

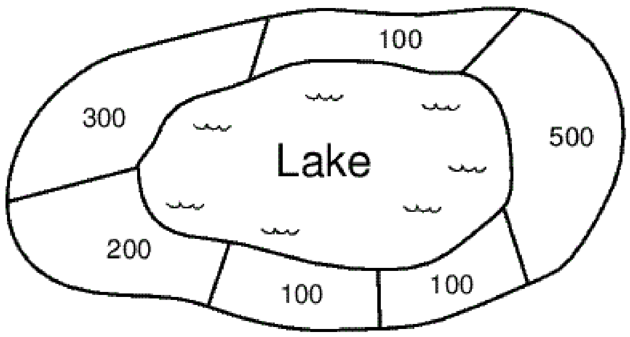
```
Yes
No

Yes
```

# 3151 - Land Division Tax

Latin America - South America - 2004/2005

PDF | PostScript | Submit | | Ranking

International Concrete Projects Company (ICPC) is a construction company which specializes in building houses for the high-end market. ICPC is planning a housing development for new homes around a lake. The houses will be built in lots of different sizes, but all lots will be on the lake shore. Additionally, every lot will have exactly two neighbors in the housing development: one to the left and one to the right.



Development plan indicating the sizes of the lots (in units of area) in the new housing development.

ICPC owns the land around the lake and needs to divide it into lots according to the housing development plan. However, the County Council has a curious regulation regarding land tax, intended to discourage the creation of small lots:

1. land can only be divided using a sequence of land divisions;
2. a land division is an operation that divides one piece of land into two pieces of land; and
3. for each land division, a land division tax must be paid.

Denoting by A the area of the largest resulting part of the division, the value of the land division tax is $A \times F$, where F is the division tax factor set yearly by the County Council. Note that due to (2), in order to divide a piece of land into N lots, N - 1 land divisions must be performed, and therefore N - 1 payments must be made to the County Council.

For example, considering the figure above, if the division tax factor is 2.5 and the first land division separates the lot of 500 units of area from the other lots, the land division tax to be paid for this first division is $2.5 \times (300 + 200 + 100 + 100 + 100)$. If the next land division separates the lot of 300 units together with its neighbor lot of 100 units, from the set of the remaining lots, an additional $2.5 \times (300 + 100)$ must be paid in taxes, and so on. Note also that some land divisions are not possible, due to (2). For example, after the first land division mentioned above, it is not possible to make a land division to separate the lot of 300 units together with the lot of 200 units from the remaining three lots, because more than two parts would result from that operation.

Given the areas of all lots around the lake and the current value of the division tax factor, you must write a program to determine the smallest total land division tax that should be paid to divide the land according to the housing development plan.

## Input

The input contains several test cases. The first line of a test case contains an integer N and a real F, indicating respectively the number of lots ($1 \le N \le 200$) and the land division tax factor (with precision of two decimal digits, $0 < F \le 5.00$). The second line of a test case contains N integers $X_i$, representing the areas of contiguous lots in the development plan ($0 < X_i \le 500$, for $1 \le i \le N$); furthermore, $X_k$ is neighbour to $X_{k+1}$ for $1 \le k \le N - 1$, and $X_N$ is neighbour to $X_1$. The end of input is indicated by $N = F = 0$.

## Output

For each test case in the input your program must produce a single line of output, containing the minimum total land division tax, as a real number with precision of two decimal digits.

## Sample Input

```
4 1.50
```

```
2 1 4 1
6 2.50
300 100 500 100 100 200
0 0
```

## Sample Output

```
13.50
4500.00
```

---

# Problem C. Cellular Automaton

| | |
|---|---|
| Input file: | `cell.in` |
| Output file: | `cell.out` |

A *cellular automaton* is a collection of cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules that describe the new state of a cell based on the states of neighboring cells. The *order of the cellular automaton* is the number of cells it contains. Cells of the automaton of order $n$ are numbered from 1 to $n$.

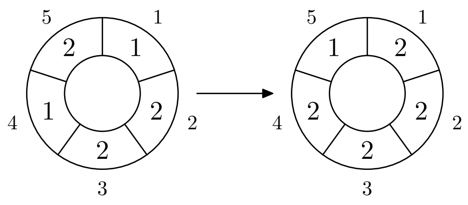
The *order of the cell* is the number of different values it may contain. Usually, values of a cell of order $m$ are considered to be integer numbers from 0 to $m-1$.

One of the most fundamental properties of a cellular automaton is the type of grid on which it is computed. In this problem we examine the special kind of cellular automaton — circular cellular automaton of order $n$ with cells of order $m$. We will denote such kind of cellular automaton as *$n$,$m$-automaton*.

A distance between cells $i$ and $j$ in $n$,$m$-automaton is defined as $\min(|i-j|, n-|i-j|)$. A *$d$-environment of a cell* is the set of cells at a distance not greater than $d$.

On each *$d$-step* values of all cells are simultaneously replaced by new values. The new value of cell $i$ after $d$-step is computed as a sum of values of cells belonging to the $d$-enviroment of the cell $i$ modulo $m$.

The following picture shows 1-step of the 5,3-automaton.



The problem is to calculate the state of the $n$,$m$-automaton after $k$ $d$-steps.

## Input

The first line of the input file contains four integer numbers $n$, $m$, $d$, and $k$ ($1 \le n \le 500$, $1 \le m \le 1\,000\,000$, $0 \le d < \frac{n}{2}$, $1 \le k \le 10\,000\,000$). The second line contains $n$ integer numbers from 0 to $m-1$ — initial values of the automaton's cells.

## Output

Output the values of the $n$,$m$-automaton's cells after $k$ $d$-steps.

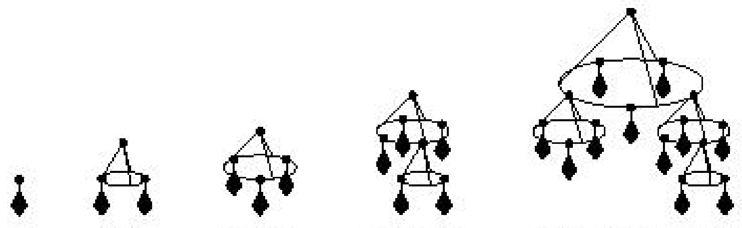
## Sample input and output

| `cell.in` | `cell.out` |
|---|---|
| `5 3 1 1`<br>`1 2 2 1 2` | `2 2 2 2 1` |
| `5 3 1 10`<br>`1 2 2 1 2` | `2 0 0 2 2` |

# 3215 – Chandelier

**Europe – Northeastern Europe & Russian Republic – 2004/2005**

Lamps–O–Matic company assembles very large chandeliers. A chandelier consists of multiple levels. On the first level crystal pendants are attached to the rings. Assembled rings and new pendants are attached to the rings of the next level, and so on. At the end there is a single large ring –– the complete chandelier with multiple smaller rings and pendants hanging from it. A special–purpose robot assembles chandeliers. It has a supply of crystal pendants and empty rings, and a stack to store elements of a chandelier during assembly. Initially the stack is empty. Robot executes a list of commands to assemble a chandelier.



On command ``a'' robot takes a new crystal pendant and places it on the top of the stack. On command ``1'' to ``9'' robot takes the corresponding number of items from the top of the stack and consecutively attaches them to the new ring. The newly assembled ring is then placed on the top of the stack. At the end of the program there is a single item on the stack –– the complete chandelier. Unfortunately, for some programs it turns out that the stack during their execution needs to store too many items at some moments. Your task is to optimize the given program, so that the overall design of the respective chandelier remains the same, but the maximal number of items on the stack during the execution is minimal. A pendant or any complex multi–level assembled ring count as a single item of the stack. The design of a chandelier is considered to be the same if each ring contains the same items in the same order. Since rings are circular it does not matter what item is on the top of the stack when the robot receives a command to assemble a new ring, but the relative order of the items on the stack is important. For example, if the robot receives command ``4'' when items $\langle i_1, i_2, i_3, i_4 \rangle$ are on the top of the stack in this order ($i_1$ being the topmost), then the same ring is also assembled if these items are arranged on the stack in the following ways: $\langle i_2, i_3, i_4, i_1 \rangle$, or $\langle i_3, i_4, i_1, i_2 \rangle$, or $\langle i_4, i_1, i_2, i_3 \rangle$.

## Input

Input file contains several test cases. Each of them consists of a single line with a valid program for the robot. The program consists of at most 10 000 characters.

## Output

For each test case, print two output lines. On the first line write the minimal required stack capacity (number of items it can hold) to assemble the chandelier. On the second line write some program for the assembly robot that uses stack of this capacity and results in the same chandelier.

## Sample Input

```
aaaaa3aaa2aaa45
```

## Sample Output

```
6
aaa3aaa2aaa4aa5
```

Northeastern Europe & Russian Republic 2004−2005

The 2001 25th Annual **acm** International Collegiate
**Programming Contest World Finals**
**sponsored by IBM**

# Problem B
## Say Cheese
### Input: cheese.in

Once upon a time, in a giant piece of cheese, there lived a cheese mite named Amelia Cheese Mite. Amelia should have been truly happy because she was surrounded by more delicious cheese than she could ever eat. Nevertheless, she felt that something was missing from her life.

One morning, her dreams about cheese were interrupted by a noise she had never heard before. But she immediately realized what it was — the sound of a male cheese mite, gnawing in the same piece of cheese! (Determining the gender of a cheese mite just by the sound of its gnawing is by no means easy, but all cheese mites can do it. That's because their parents could.)

Nothing could stop Amelia now. She had to meet that other mite as soon as possible. Therefore she had to find the fastest way to get to the other mite. Amelia can gnaw through one millimeter of cheese in ten seconds. But it turns out that the direct way to the other mite might not be the fastest one. The cheese that Amelia lives in is full of holes. These holes, which are bubbles of air trapped in the cheese, are spherical for the most part. But occasionally these spherical holes overlap, creating compound holes of all kinds of shapes. Passing through a hole in the cheese takes Amelia essentially zero time, since she can fly from one end to the other instantly. So it might be useful to travel through holes to get to the other mite quickly.

For this problem, you have to write a program that, given the locations of both mites and the holes in the cheese, determines the minimal time it takes Amelia to reach the other mite. For the purposes of this problem, you can assume that the cheese is infinitely large. This is because the cheese is so large that it never pays for Amelia to leave the cheese to reach the other mite (especially since cheese-mite eaters might eat her). You can also assume that the other mite is eagerly anticipating Amelia's arrival and will not move while Amelia is underway.

## Input

The input file contains descriptions of several cheese mite test cases. Each test case starts with a line containing a single integer $n$ ($0 \leq n \leq 100$), the number of holes in the cheese. This is followed by $n$ lines containing four integers $x_i, y_i, z_i, r_i$ each. These describe the centers $(x_i, y_i, z_i)$ and radii $r_i$ ($r_i > 0$) of the holes. All values here (and in the following) are given in millimeters.

The description concludes with two lines containing three integers each. The first line contains the values $x_A, y_A, z_A$, giving Amelia's position in the cheese, the second line containing $x_O, y_O, z_O$, gives the position of the other mite.

The input file is terminated by a line containing the number –1.

## Output

For each test case, print one line of output, following the format of the sample output. First print the number of the test case (starting with 1). Then print the minimum time in seconds it takes Amelia to reach the other mite, rounded to the closest integer. The input will be such that the rounding is unambiguous.

**Sample Input**

```
1
20 20 20 1
0 0 0
0 0 10
1
5 0 0 4
0 0 0
10 0 0
-1
```

**Output for the Sample Input**

```
Cheese 1: Travel time = 100 sec
Cheese 2: Travel time = 20 sec
```