ACM International Collegiate Programming Contest

ARAB COLLEGIATE programming contest

2010 Thirteenth
Arab Collegiate Programming Contest

Lebanese American University
Beirut, Lebanon
November 2010

The problem set is made of <u>11 numbered pages.</u>
(This edition includes all major clarifications that were announced during the contest)

[A] What's Next?

Program:	`next.(c|cpp|java)`
Input:	`next.in`
Balloon Color:	`Dark Green`

Description

According to Wikipedia, *an arithmetic progression (AP)* is a sequence of numbers such that the difference of any two successive members of the sequence is a constant. For instance, the sequence 3, 5, 7, 9, 11, 13, ... is an arithmetic progression with common difference 2. For this problem, we will limit ourselves to arithmetic progression whose common difference is a non-zero integer.

On the other hand, a *geometric progression (GP)* is a sequence of numbers where each term after the first is found by multiplying the previous one by a fixed non-zero number called the common ratio. For example, the sequence 2, 6, 18, 54, ... is a geometric progression with common ratio 3. For this problem, we will limit ourselves to geometric progression whose common ratio is a non-zero integer.

Given three successive members of a sequence, you need to determine the type of the progression and the next successive member.

Input Format

Your program will be tested on one or more test cases. Each case is specified on a single line with three integers ($-10,000 < a_1, a_2, a_3 < 10,000$) where a_1, a_2, and a_3 are distinct.

The last case is followed by a line with three zeros.

Output Format

For each test case, you program must print a single line of the form:

```
XX␣v
```

where `XX` is either `AP` or `GP` depending if the given progression is an Arithmetic or Geometric Progression. `v` is the next member of the given sequence. All input cases are guaranteed to be either an arithmetic or geometric progressions.

Sample Input/Output

next.in
```
4 7 10
2 6 18
0 0 0
```

OUTPUT
```
AP 13
GP 54
```

[B] Sum the Square

Program:	`distance.(c|cpp|java)`
Input:	`distance.in`
Balloon Color:	`Yellow`

Description

Take any positive number, find the sum of the squares of its digits, repeat! You'll end up with an infinite sequence with an interesting property that we would like to investigate further. Starting with the number 5, the sequence is:

$$(5, 25, 29, 85, 89, 145, 42, 20, 4, 16, 37, 58, \ldots)$$

The interesting part is in what comes after 58: $5^2 + 8^2 = 89$ which is a number that's already been seen in the sequence. In other words, after 58, the sequence will fall into the repeating cycle: $89, 145, 42, 20, 4, 16, 37, 58$.

What's amazing is that this cycle will appear for many other numbers: 3, 18, 36, and 64 just to name a few. (see figure on the following page.)

For some numbers, the sequence will fall into another repeating cycle by reaching 1. (see second figure on the following page) For example, starting with 19, you'll end up with the sequence:

$$(19, 82, 68, 100, 1, \ldots)$$

And that's about it. Any number you choose will end up falling into a repeating cycle: Either the $89, 145, \ldots$ cycle or the $1, \ldots$ cycle.

Given two numbers, your objective is to generate as few numbers in their sequences for the two sequences to intersect at one common number. For example, given 61 and 29, we can achieve what's required by generating the sequences: $(61, 37, 58, \mathbf{89})$ and $(29, 85, \mathbf{89})$. Similarly, for 19 and 100, the sequences would be $(19, 82, 68, \mathbf{100})$ and $(\mathbf{100})$.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line having two integers $(0 < A, B < 10^9)$.

The last case is followed by a dummy line made of two zeros.

Output Format

For each test case, print the following line:

```
A␣B␣S
```

Where A, B are as in the input and S is the (minimum) sum of the lengths of the two sequences. If the sequences starting at A and B do not intersect, then $S = 0$.

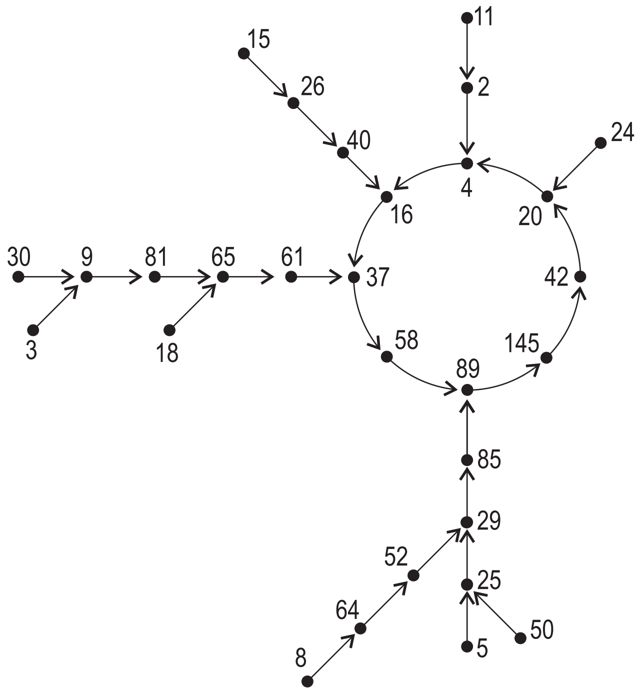
Sample Input/Output

distance.in

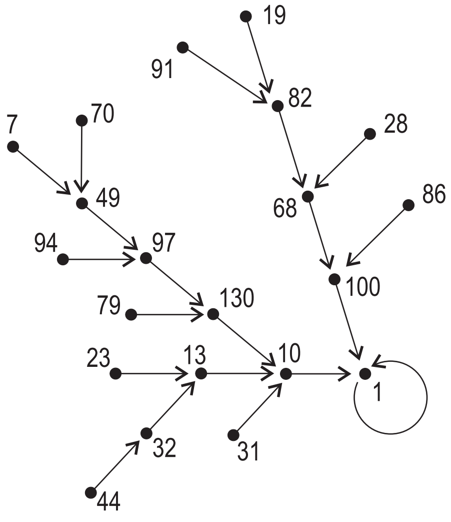
```
89 89
19 100
61 19
0 0
```

OUTPUT

```
89 89 2
19 100 5
61 19 0
```



Few numbers falling into the $89, 145, 42, 20, 4, 16, 37, 58$ cycle



Few numbers falling into the 1 cycle.

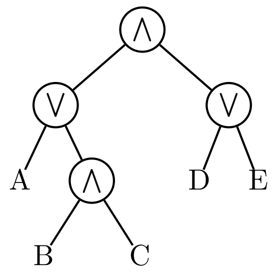
[C] Normalized Form

Program:	`tree.(c|cpp|java)`
Input:	`tree.in`
Balloon Color:	`Blue`

Description

As you most probably know, any boolean expression can be expressed in either a *disjunctive normal form* or a *conjunctive normal form.* In a disjunctive normal form, a boolean expression is written as a disjunct (logical or) of one-or more sub-expressions where each of these sub-expressions is written in a conjunctive normal form. Similarly, an expression written in a conjunctive normal form is a conjunct (logical and) of sub-expressions each written in a disjunctive normal form.

An AND/OR tree is a tree-like graphical-representation of boolean expressions written as either conjunctive- or disjunctive-normal form. Since the sub-expressions of a normalized form alternate in being either disjunctive or conjunctive forms, you'd expect the sub-trees on an AND/OR tree to alternate in being AND- or OR- trees depending on the sub-tree's depth-level. The example on the right illustrates this observation for the boolean expression $(A \bigvee (B \bigwedge C)) \bigwedge (D \bigvee E)$ where the trees in the 1st (top-most) and 3rd levels are AND-trees.



Write a program that evaluates a given and/or tree.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on exactly one line (which is no longer than 32,000 characters) of the form:

$$(E_1\ E_2\ \ldots\ E_n)$$

where $n > 0$ and E_i is either `T` for true, `F` for false, or a sub-expression using the same format. The trees at the deepest level are AND-trees. The last test case is followed by a dummy line made of ().

Output Format

For each test case, print the following line:

```
k.␣E
```

Where `k` is the test case number (starting at one,) and `E` is either `true` or `false` depending on the value of the expression in that test case.

Sample Input/Output

tree.in
```
((F(TF))(TF))
(TFT)
((TFT)T)
()
```

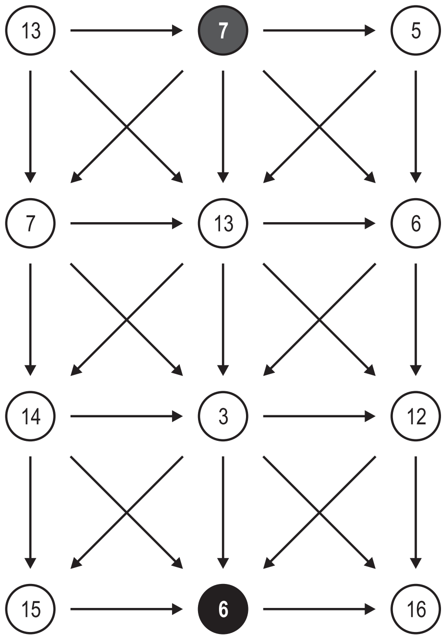
OUTPUT
```
1. false
2. false
3. true
```

[D] Tri graphs

Program:	`tri.(c|cpp|java)`
Input:	`tri.in`
Balloon Color:	`Orange`

Description

Here's a simple graph problem: Find the shortest path from the top-middle vertex to the bottom-middle vertex in a given tri-graph. A tri-graph is an acyclic graph of ($N \geq 2$) rows and exactly 3 columns. Unlike regular graphs, the costs in a tri-graph are associated with the vertices rather than the edges. So, considering the example on the right with $N = 4$, the cost of going straight down from the top to bottom along the middle vertices is $7 + 13 + 3 + 6 = 29$. The layout of the directional edges in the graph are always the same as illustrated in the figure.



Input Format

Your program will be tested on one or more test cases. Each test case is specified using $N + 1$ lines where the first line specifies a single integer ($2 \leq N \leq 100,000$) denoting the number of rows in the graph. N lines follow, each specifying three integers representing the cost of the vertices on the i^{th} row from left to right. The square of each cost value is less than 1,000,000.
The last case is followed by a line with a single zero.

Output Format

For each test case, print the following line:

```
k.␣n
```

Where `k` is the test case number (starting at one,) and `n` is the least cost to go from the top-middle vertex to the bottom-middle vertex.

Sample Input/Output

tri.in
```
4
13 7 5
7 13 6
14 3 12
15 6 16
0
```

OUTPUT
```
1. 22
```

[E] Sometimes, a penalty is good!

Program:	`wakawaka.(c|cpp|java)`
Input:	`wakawaka.in`
Balloon Color:	Red

Description

FIFA is considering a few changes to the way it organizes the Football World Cup. Currently, 32 teams compete for the World Title in two stages. During the first stage, known as the groups stage, the 32 teams are split evenly into 8 groups. Every team in the group plays 3 games, one against each team in their own group. Teams are then ranked within their group according to some points system. During the second (and final) stage, the top two teams from each group advance to the knockout stage where eight games are played to determine eight winners who would then play four games to determine four winners, then two games to determine the two winners who would then play the final game to determine the world champion. Needless to say, for the knockout stage to work, the number of teams in that stage has to be a power of two.

FIFA is considering adding more groups, adding more teams to groups, and possibly changing the number of teams advancing from each group to the knockout stage. In addition, FIFA is considering having certain teams (previous champion, host country, etc.) advance to the knockout stage directly (without having to play in the groups stage.) But FIFA needs to know how many games will be played if any of these changes are applied. Please help them!

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line made of 4 natural numbers with the following format:

```
G T A D
```

Where ($G > 0$) is the number of groups; T is the number of teams in each group; A is the number of teams advancing from each group to the knockout stage; and D is the number of teams directly advancing to the knockout stage without going through the groups stage. Note that ($0 < A \leq T$) and that the four numbers in the input are no larger than 2^{16}.

If the total number of teams in the knockout stage is not a power of two, your program must increase them to the closest power of two.

The last test case is followed by a dummy line made of four -1's.

Output Format

For each test case, print:

```
G*A/T+D=X+Y
```

where `G`, `A`, `T`, and `D` are as in the input, `X` is the total number of games, and `Y` is the number of teams your program determined it must add.

Sample Input/Output

wakawaka.in

```
8 4 2 0
8 4 2 1
-1 -1 -1 -1
```

OUTPUT

```
8*2/4+0=63+0
8*2/4+1=79+15
```

[F] World of cubes

Program:	`cube.(c|cpp|java)`
Input:	`cube.in`
Balloon Color:	`Purple`

Description

You've been assigned the task of programming a new video game called the World of Cubes. The game is played inside a box-shaped object (mathematically speaking: a rectangular cuboid) which we'll call *the hall.* Within the hall, the player is assigned N positions, called *focal points.* The player must build N cubes, all parallel to the axes, and each is centered around one of the focal points. The mission is for the N cubes to completely fill the hall. It is acceptable for the cubes to overlap and even to extend beyond the hall. The only restriction is that all the cubes must be of equal side-length. And we would like you to compute the minimum such length.

Input Format

Your program will be tested on one or more test cases. Each test case is specified using $N+1$ lines. The first line specifies four integers: $(1 \leq N \leq 50)$ is the number of cubes and $(1 \leq X, Y, Z \leq 10^9)$ are the dimensions of the hall. One corner of the hall is at the origin $(0, 0, 0)$, with the opposing corner at (X, Y, Z).

N lines follow, each specifying the coordinate of a focal point using three integers: $(0 \leq x \leq X)$, $(0 \leq y \leq Y)$, and $(0 \leq z \leq Z)$.

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

Output Format

For each test case, print the following line:

```
k.␣D
```

Where `k` is the test case number (starting at one,) and `D` is an integer which is the smallest length of the cube edges so that the N cubes would completely fill the hall.

Sample Input/Output

cube.in

```
2 4 4 8
2 2 2
2 2 6
2 4 4 8
2 2 2
2 2 5
0 0 0 0
```

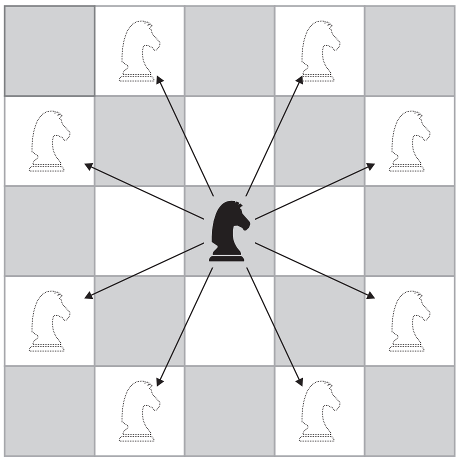
OUTPUT

```
1. 4
2. 6
```

[G] A Knights' Tale

Program:	`knights.(c|cpp|java)`
Input:	`knights.in`
Balloon Color:	`Black`

Description



Imagine a chess board that extends indefinitely in both horizontal and vertical directions. N identical knights are placed on this board, each in a different square. N different squares are specially marked, which we will call the *target squares,* which could be different from where the knights are initially at. We would like you to determine the minimum number of knight-steps needed so that each of the target squares is occupied by one of the knights.
As illustrated in the figure, a knight moves using the normal "L" move (1 square in one dimension and 2 squares in the other dimension.) For this problem, it is possible for more than one knight to occupy the same square while trying to reach its final destination as long as each knight ends up in a different target square.

Input Format

Your program will be tested on one or more test cases. Each test case is specified using $2N+1$ lines. The first line specifies $(1 \leq N \leq 15)$ which is the number of knights (or targets.) The following N lines each specifies the position of a knight by specifying two integers representing the x and y location. The remaining N lines each specifies the position of a target square again by specifying two integers representing the x and y location. All coordinates are 32-bit signed integers.

The last case is followed by a line with a single zero.

Output Format

For each test case, print the following line:

```
k.␣m
```

Where `k` is the test case number (starting at one,) and `m` is the minimum number of moves.

Sample Input/Output

knights.in
```
2
3 5
6 5
5 3
7 3
0
```

OUTPUT
```
1. 3
```

[H] Jumping Beans

Program:	`beans.(c|cpp|java)`
Input:	`beans.in`
Balloon Color:	`White`

Description

N jumping beans are standing in a line. At each second, a bean jumps. Your assignment is to figure the final position of the beans after a given number of seconds.

To make the explanation easier, let's assign a unique letter to each bean, and for simplicity, let's assume the beans are initially standing in order: `A`, `B`, `C`, etc. To simplify even further, let's assume $N = 4$, so initially the beans are standing in the order `ABCD`.

At the first second, `A` jumps, swapping its place with `B`. Now the order is `BACD`. At the second second, it's `B`'s turn, but this time swapping its place with `A`, then `C`, resulting in the standing order `ACBD`. More formally: at second s, the left most bean that has jumped the least number of times will do the swap s times, each time swapping its place with the bean on its right. Note that when the right-most bean swaps, it moves to the left-most position, pushing everybody else one place to the right.

So, continuing with the previous example, and starting with the arrangement `ACBD`, it is bean `C`'s turn, since it is the left-most bean that has jumped the least amount of times. Being at the third second, `C` will swap three times, first resulting in `ABCD`, then `ABDC`, and then `CABD`. At the fourth second, it's bean `D`'s turn to jump. At the fifth second, and since all the four beans have jumped exactly once, the bean that will jump is the bean standing at the left-most position.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line specifying an integer T and a string S where $(0 < T < 10^9)$ is the number of seconds and S is the initial arrangement of the beans. S is a non empty string made of different upper-case letters (`'A'`...`'Z'`).

The last test case is followed by a line having a single 0.

Output Format

For each test case, print the following line:

```
k.␣S
```

Where `k` is the test case number (starting at one,) and `S` is the arrangement of the beans after jumping for `T` seconds.

Sample Input/Output

beans.in

```
3 ABCD
13 ACM
0
```

OUTPUT

```
1. CABD
2. CAM
```

[I] The Cyber Traveling Salesman

Program:	`moon.(c|cpp|java)`
Input:	`moon.in`
Balloon Color:	`Pink`

Description

In light of the exploding population on earth, a number of cities are being constructed on the moon. We would like you to assist in determining the best road system for these cities. Considering the high cost associated with building roads on the moon, all what is required is for the roads to form a cycle starting from the city that appears first in the input, passing through all other cities exactly once (but in any arbitrary order,) and then ending back to the first. (Yes, this problem is a variation of the traveling-salesman problem.)

You are given the cost of building a road between each pair of cities. Roads are one-way, but the cost for building a road from city i to j is the same as the cost of building from city j to i. When roads intersect at a location that is not a city, you must account for the cost of constructing bypassing bridges. Constructing a bypass system costs $k * (k - 1) * C/2$ where k is the number of roads intersecting at that location, and C is a given constant.

Note that the cities are laid out so that no three cities fall on the same straight line.

Input Format

Your program will be tested on one or more test cases. Each test case is specified using $2 * N + 1$ lines. The first line specifies two integers: $(2 < N < 9)$ is the number of cities and $(0 < C \leq 1,000,000)$ is the coefficient used in determining the cost of building bridges.

Following the first line, the Cartesian coordinates of the cities are specified in order. Each city is specified on a separate line made of two integers: x_i and y_i where $(-1,000 \leq x_i, y_i \leq 1,000)$. No two cities are located at the same (x, y) location.

The last N lines of a test case specify an $N * N$ matrix representing the cost of building a road between any two cities. The matrix is specified using N lines, each with N integers in a row-major format. The j^{th} value on the i^{th} row, denoted as c_{ij} is the cost of building a road from city-i to city-j where $(0 < c_{ij} \leq 10^6)$ and $(c_{ij} = c_{ji})$ and $(c_{ii} = 0)$.

The last case is followed by a line having two zeros.

Output Format

For each test case, print the following line:

```
k.␣M
```

Where `k` is the test case number (starting at one,) and `M` is minimum cost needed to build the road system.

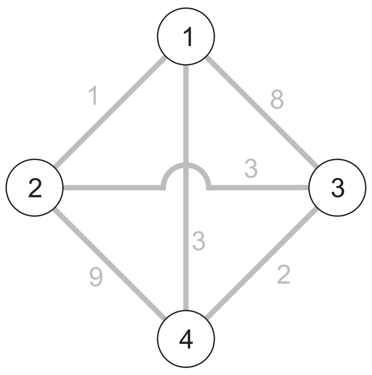
Sample Input/Output

moon.in

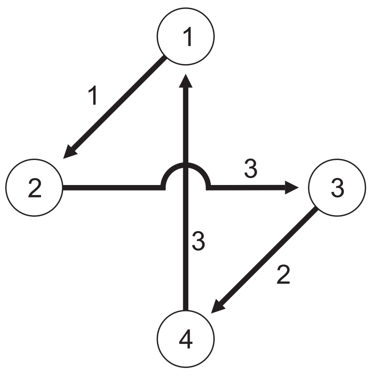
```
4 1
1 2
0 1
2 1
1 0
0 1 8 3
1 0 3 9
8 3 0 2
3 9 2 0
4 100
1 2
0 1
2 1
1 0
0 1 8 3
1 0 3 9
8 3 0 2
3 9 2 0
0 0
```

OUTPUT

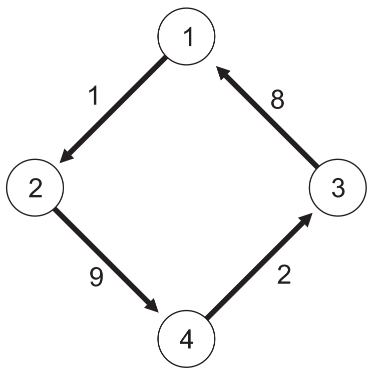
```
1. 10
2. 20
```



Input data for both cases



A solution for case #1



A solution for case #2