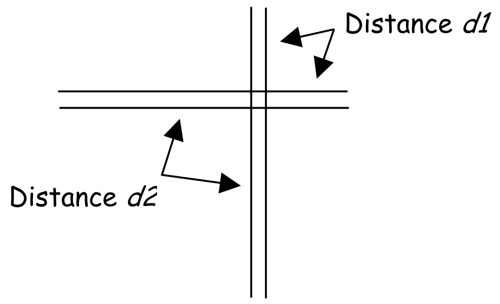
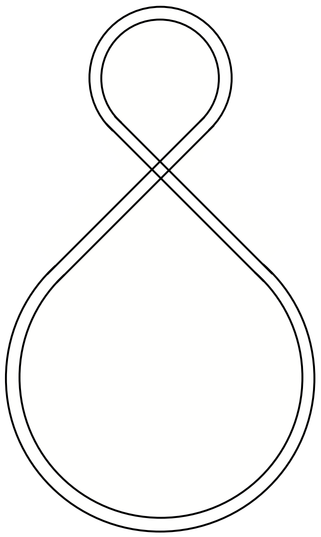
Problem 1: Prime Tracks

For her birthday, Colleen was given a section of track in order to start a toy train set. The section was a railroad crossing of exactly 90° but with different lengths, as shown in the figure on the left below.





The distances *d1* and *d2* are specified in integral numbers of millimeters.

Since this is the only piece of track she owned, she decided to purchase additional track in order to construct a figure eight, as shown in the figure on the right.

The top and bottom regions are 270° portions of a circle, each with a constant radius, so that the arcs exactly line up with each respective portion of the crossing.

Colleen goes to the toy store, Mathematical Marvels, to purchase track. "The reason the store is called Mathematical Marvels," explains the owner, "is that we only sell flexible track, in sections that happen to be lengths that are prime numbers." To make matters worse, there is a track shortage! The store only carries one track piece with each prime number length between 3 and 997. The store charges by the piece, not the length, of track, and Colleen only has enough money for a maximum of five sections for the top oval, and five sections for the bottom oval.

You volunteer to help Colleen by writing a program to minimize the cost of purchasing the required train track. To do this you will determine the smallest number of primes, and their values, that give the lengths of track sections that must be purchased in order to complete the figure eight. There may be more than one correct set of primes for each arc, but the minimal cardinality for each set of sections is desired. Since the track sections are flexible, a tolerance of ± 1 mm on each arc distance is allowed.

The input to the program should be the two distance measurements D1 and D2. The output should be a list of prime numbers necessary for the top loop and a list of prime numbers necessary for the bottom loop. Because of the track shortage these primes must all be unique.

Input

The input will contain multiple cases. For each case, the input will consist of two integers giving the lengths *d1* and *d2* as described above.

There will be at least one solution for each input case.

Input for the last case will be followed by a pair of zeroes.

Output

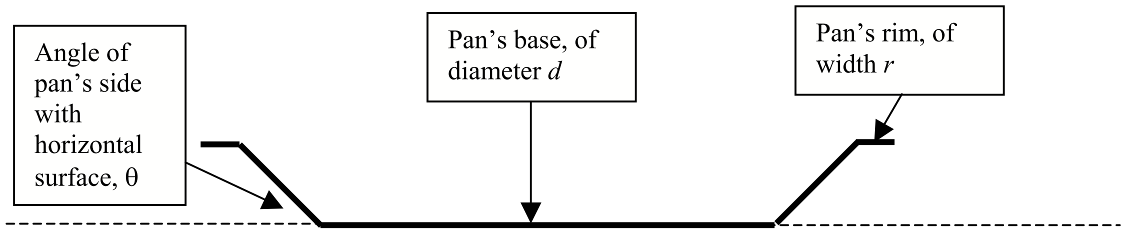
For each case, display the case number (they start with 1 and increase sequentially). Then, on the next line, display the number of segments and the prime number lengths of track (in ascending order) to be used in constructing the upper region of the figure 8. Similarly, on the third line, information about the track used to construct the lower region. Separate the output for consecutive cases with a blank line. Your output should appear similar to that shown in the sample below.

Sample Input	Output for the Sample Input

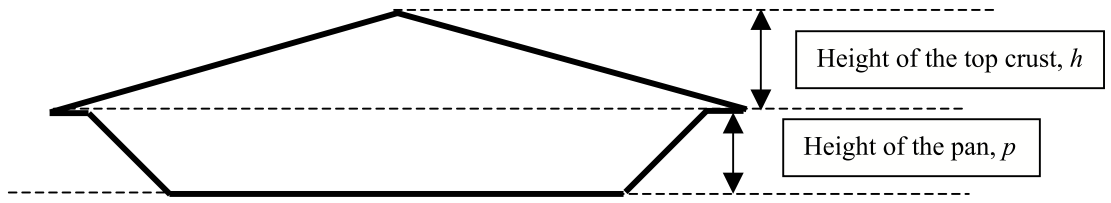
```
301 501
210 435
0 0
```
	```
	Case 1
	   4 segments: 3, 383, 977, 997
	   2 segments: 499, 919

	Case 2
	   3 segments: 3, 19, 967
	   3 segments: 61, 991, 997
	```

Problem 2: Pies

Mrs. Smith bakes apple pies. Because she makes quite a few, she wishes to optimize the process by knowing exactly how much dough is required for each pie. Her pies are baked in tins which have negligible thickness but which are of this general shape (side view):


Angle of pan's side with horizontal surface, θ
Pan's base, of diameter *d*
Pan's rim, of width *r*

Two pieces of dough are required for each pie. One piece of dough is placed in the bottom of the pan, up the inside, and out to the outside edge of the rim. The second piece of dough goes over the top of the pie, from the outside of the rim up to a point in the center of the pie forming a perfect cone. For clarity, note that the dough is two layers thick at the very outside edge of the rim.



Given values for θ, *d*, *r*, *p*, and *h* (as identified in the preceding figures), determine the volume of dough required for each pie, in cubic inches, by first calculating the necessary surface areas and then considering the dough to be 1/8 inches thick.

Input

The input will contain multiple cases. For each case, the input will consist of a line containing real numbers for θ, *d*, *r*, *p*, and *h*. Since these are real pies, θ will always be in the range 10 to 80 degrees, and the values for *d*, *r*, *p*, and *h* will be positive and less than or equal to 16 inches.

Input for the last case will be followed by a line containing a single –1.

Output

For each case, display the case number (they start with 1 and increase sequentially), and the total volume of crust required, in cubic inches, with three fractional digits. Separate the output for consecutive cases with a blank line. Your output should appear similar to that shown in the sample below.

Sample Input	Output for the Sample Input

`45.0 9.0 0.5 2.0 1.0`
`40.0 10.0 0.5 2.5 1.25`
`-1`
	`Case 1: 21.991 cubic inches`
	
	`Case 2: 29.954 cubic inches`

Problem 3: Segment Life

Decimal numbers can be displayed using one or more seven-segment displays. The segments are arranged so that illuminating selected segments yields a pattern corresponding to a decimal digit. The patterns used in this problem are show below. There are three horizontal segments (each illustrated by two hyphens), and seven vertical segments (each illustrated by a vertical stroke).

```
  --            --     --            --     --            --     --  
 |  |      |      |      |   |  |   |      |           |   |  |   |  |
                --     --     --     --     --            --     --  
 |  |      |   |         |      |      |   |  |        |   |  |      |
  --            --     --            --     --                --      
  0       1     2      3      4      5      6      7      8      9
```

Each of the segments in a single display has an expected lifetime specified by the manufacturer. This lifetime is the minimum cumulative time each segment is expected to be capable of being illuminated, independent of how frequently it is turned on or off.

In normal use, a display doesn't always illuminate the same segments. That is, different digits are displayed with different frequencies. As a result, some segments might be illuminated longer than others, and thus the actual length of time the display can be expected to display the proper results is dependent on the values it displays.

Given the values to be displayed, their probability of appearance, and the lifetime specified by the manufacturer, you are to determine the minimum time the display can be expected to display the proper values.

Multiple single-digit displays may be required, the actual number of such depending on the largest value to be displayed. Values are displayed with leading zeroes, if necessary, to use all single-digit displays. The displays are entirely blank (no segments illuminated) if no value is displayed.

Examples

As a simple example, suppose the manufacturer specifies a particular display as having a 100 hour lifetime. If only the digits 0 and 1 are displayed, each with a 25 percent probability, then the display will require only a single digit, and it will be blank 50 percent of the time. During the remaining time, the rightmost two vertical segments will be illuminated, since they are "on" during the display of 0 or 1. Given these conditions, the display will correctly operate for at least 200 hours.

Suppose the values to be displayed are 0, 1, 10, and 11, each with a probability of 25 percent. If the display has the same manufacturer-specified lifetime, then two single-digit display units will be required, and the display will correctly operate for at least 100 hours.

Finally, assume arbitrary decimal digits are to be displayed (with equal probability) 50 percent of the time. With a segment lifetime of 50 hours, the display can be expected to operate correctly for at least 111.11 hours. This is because no segment is illuminated in more than 90 percent of the digits displayed.

Input

For this problem there are multiple cases. The input for each case begins with two integers. The first of these is the manufacturer-specified lifetime of a display, in hours; the second is the number of ranges of values to be displayed. For each of these ranges there follows a group of three integers that give the lowest and highest values in the range, and the probability that one of these values will appear. The largest value will never require a display with more than six digits. Input for the last case is followed by a single integer –1.

Output

For each case, display the case number (starting with 1) and the minimum time (with two fractional digits) the display will operate correctly. Leave a single blank line between the output for consecutive cases.

Sample Input

```
100 1
    0  1  50

100 4
    0  0  25
    1  1  25
   10 10  25
   11 11  25

100 2
    0  1  50
   10 11  50

50 1
    0  9  50

-1
```

Output for the Sample Input

```
Case 1: 200.00 hours

Case 2: 100.00 hours

Case 3: 100.00 hours

Case 4: 111.11 hours
```

Problem 4: Grade Dropping

Welcome to Discrete Math I. Forty percent of your final grade will be based on your homework assignments. Since everyone can have a bad day from time to time, you will be allowed to not count up to three of these assignments, but you have to choose which assignments to drop. If all assignments counted equally (that is, the raw scores were simply averaged together) the choice would be easy — drop the assignments with the lowest scores. However, each assignment may have a different maximum score. The final homework grade will be the percentage ratio of your total score to the maximum possible score for the retained assignments.

Write a program that, given a list of assignment results, will calculate the best homework percentage grade after dropping zero, one, two, and three of the assignments.

Example

As an example, consider that you have received the following scores on each of seven assignments, with the maximum scores as show:

Assignment Number	Score	Maximum Possible
1	41	42
2	22	64
3	2	26
4	11	44
5	24	27
6	26	70
7	4	30

The homework grade without dropping any assignments is calculated as follows:

(41 + 22 + 2 + 11 + 24 + 26 + 4) / (42 + 64 + 26 + 44 + 27 + 70 + 30) = 42.9%

The best result possible for dropping only one assignment is to drop assignment 3:

(41 + 22 + 11 + 24 + 26 + 4) / (42 + 64 + 44 + 27 + 70 + 30) =46.2%

Similarly it is possible to choose the second and third assignments to drop to yield the best possible final average.

Input

There may be multiple cases. The input for each case is a series of between 4 and 30 pairs of integers terminated by a –1. The first integer in each pair is your score on an assignment; the second integer is the maximum possible score on that assignment. Each assignment has a maximum possible score between 1 and 100 points, and no assignment will ever receive a negative score.

Input for the last case is followed by a single integer –1.

Output

For each case, display the case number (starting with 1) and the maximum possible homework grade (with two fractional digits) that could be obtained by dropping zero, one, two and three assignments. Make your output appear similar to that shown in the samples below. Leave a single blank line between the output for consecutive cases.

Sample Input

```
41 42 22
64 2 26 11 44
24 27
26 70 4
30 -1

-1
```

Output for the Sample Input

```
Case 1:
   Dropping no grades:  42.90
   Dropping 1 grade:    46.21
   Dropping 2 grades:   50.22
   Dropping 3 grades:   56.80
```

Problem 5: Nested Boxes

It is a common prank to give someone a gift in a large box, in which is nested a smaller box, with another smaller box inside that one, and so forth, until the smallest box — nested within all those other boxes — contains the gift. Given a set of boxes of various sizes, your problem is to find the size (cardinality) of the largest subset of boxes that can be used to create such a nested arrangement. If no boxes can be nested, then the size of the subset is just 1.

Naturally, each box in the set from which you can choose has three dimensions. Any box can be rotated, if desired, if that would enable it to fit inside another box. For our purposes, a box A can fit inside a box B if each dimension of box A is strictly less than the corresponding dimension of box B.

Example

For example, suppose box A has dimensions $12 \times 20 \times 60$ and box B has dimensions $42 \times 18 \times 10$. If we rotate box B appropriately — so the dimensions are $10 \times 18 \times 42$, then we will be able to nest it inside box A. However, if box B had dimensions $13 \times 11 \times 58$, then no rotations would allow it to fit inside box A.

Input

There may be multiple cases. The input for each case begins with a line containing a single integer, *N*, that specifies the number of boxes in the set from which you are allowed to choose. This line will be followed by *N* more lines, each containing three positive non-zero integers giving the dimensions of a box. N will be no larger than 500, and no box will have a dimension larger than 999.

Input for the last case is followed by a single integer –1.

Output

For each case, display the case number (starting with 1) and the maximum number of boxes selected from the set that can be nested as described. Make your output appear similar to that shown in the samples below. Leave a single blank line between the output for consecutive cases.

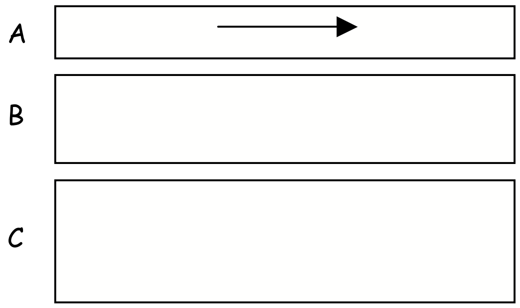
Sample Input

```
5
145 472 812
827 133 549
381 371 900
271 389 128
718 217 491
4
432 123 139
942 844 783
481 487 577
677 581 701
-1
```

Output for the Sample Input

```
Case 1: 2 boxes

Case 2: 4 boxes
```

Problem 6: Conveyor Belts



The package sorting area of a shipping company consists of an incoming bin of packages, and three conveyor belts, "A", "B", and "C", of varying widths. Each conveyor belt moves to the right. Packages fall off the end of the conveyor and into some other sorting bin. Suppose that the conveyors are of widths 10", 20", and 30" as in the diagram shown to the right.

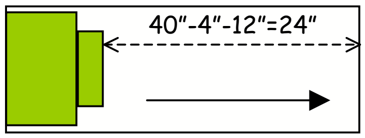
All the packages are "simple six sided boxes" with three dimensions. The person doing the sorting places each box on the appropriate conveyor. The proper conveyor to use is the narrowest one which is still wide enough to hold the package. We wish to use as little conveyor length as possible. All packages will fit on at least one conveyor, and (for example) a 20" conveyor would hold up to and including a 20" distance on one side of the package. Packages are placed on the conveyor with no space between them. In order to select the conveyor, you need to consider all dimensions of the package.

Examples

Suppose a certain box is 15" x 21" x 4". The sorting process first considers using the minimal conveyor length, so we desire to put the box on a conveyor such that the 4" side is traveling in the conveyor direction to the right. We can thus place the box on a 4" x 21" side so that it is 15" high, or we can place it on a 4" x 15" side and the package will stand 21" high. Since we want to use the narrowest possible conveyor, we select conveyor "B" since the 20" width of the conveyor will handle the 15" dimension of the box ("A" is too narrow, "C" is excessively wide). At this point, conveyor "B" moves to the right 4" to accommodate the box, and the box is placed on the far left end of the belt.

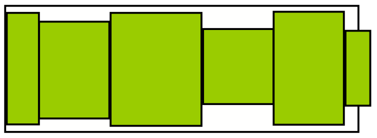
Another box arrives which is 12" x 24" x 19". We use the 12" dimension for the linear distance, and can either stand the box on a 12" x 19" surface so it is 24" high, or on a 12" x 24" surface so it is 19" high. We choose conveyor "B" since 19" <= 20". Conveyor "B" moves to the right by 12" to accommodate the box, and we place it on the belt.

After the two boxes described above have been processed, conveyor "B" will look like this, with the first box to the right of the second box. Note that 24" of the conveyor belt is still unused.



As the belts move from left to right, packages will eventually fall off the end. The conveyor belts are 40" long. A package must be more than 50% over the right edge in order to fall off. For example, a box which occupies 5" of belt space will fall off when 3" has passed the end of the belt, not 2". Since package dimensions are given as integral numbers of inches, there is no need to check for 2.5". Similarly, a box taking up 4" of conveyor space falls off at 3" but stays on at 2".

Since there is 24" of conveyor remaining in this case, the first box will fall off when the belt has moved more than 26" further (recall that the first box is 4" on this dimension). The diagram shows the first box about to fall off the conveyor.



Input

There may be multiple cases. The input for each case begins with a line containing three integers that specify the widths of the conveyor belts, in increasing order. That is, conveyor "A" is the narrowest, and conveyor "C" is the widest. No two conveyor belts will have the same width. This line will be followed by additional lines, one for each box, in the order they are placed on the conveyor belts. Each of these lines contains three positive non-zero integers giving the dimensions of a box. The first of these lines gives data for box 1, the second for box 2, and so forth. The last box in each case will be followed by a line containing three integers, each –1.

Input for the last case is followed by a line containing three integer zeroes.

Output

For each case, display the case number (starting with 1) on a line by itself. Then display additional lines, each of which gives the number of a box and the letter (A, B, or C) identifying the conveyor belt from which it fell. These must be in the correct order for the given input. Make your output appear similar to that shown in the samples below. Leave a single blank line between the output for consecutive cases. Note that there will be fewer lines of output than there are boxes, since some boxes will remain on the conveyors at the end of the program. Also, placing one large box on the conveyor may force more than one box (or no boxes) off the end.

Sample Input	Output for the Sample Input

10 15 20
6 8 10
4 11 8
19 8 8
5 9 12
10 10 10
8 8 8
9 9 14
-1 -1 -1
0 0 0
	Case 1:
	   Box 1 fell off conveyor A
	   Box 2 fell off conveyor A

Problem 7: Happy Birthday to You, and You, and You!

Many people recognize the "birthday paradox." It states that given a group of 23 or more randomly chosen persons, the probability that at least two of them will have the same birthday is more than 50%. Actually it isn't a paradox, but is so named because it contradicts what most people would intuitively expect.

But what is the probability that at least three persons in a group of *N* randomly chosen persons share the same birthday?

For simplicity we will assume that there are exactly 365 days in a year, and require that your answer be correct only within one tenth of a percent. For example, with 88 people in a group, the probability that at least three of these people will have the same birthday is 51.1 percent (with the result rounded to one fractional digit). Your answer could be any of 51.0 percent, 51.1 percent, or 51.2 percent and still be considered correct.

Input

There may be multiple cases. The input for each case is a line containing the integer *N*, the size of the group. *N* will be no larger than 1000.

Input for the last case is followed by a line containing -1.

Output

For each case, display the case number (starting with 1) and the probability that at least three persons in a group of size N share the same birthday. Your result should be displayed as a percentage rounded to one fractional digit. Separate the output for consecutive cases by a blank line. Your output should look very similar to that shown in the samples below.

Sample Input	Output for the Sample Input

```
23
88
-1
```
	```
	Case 1. 1.3 percent

	Case 2. 51.1 percent
	```

Problem 8: E-mail Sniffing

Your company is constructing a device for the International Criminal Protection Council ICPC). This device can be used to detect e-mail messages containing words that suggest the message relates to an illegal activity, and thus require further attention. Since e-mail is abundant, and the networks are fast, the device needs to detect these words very quickly. You have decided to construct a hash code for each word in an e-mail and then determine if that word is in a hash table of suspect words. To make the lookup fast, you have decided to use a perfect hash function where each suspect word maps to a unique location in the table.

A perfect hash function maps its input directly to a fully-occupied table. You need to construct the perfect hash function from the list of suspect words. The hash function is of the form $\lfloor C / w \rfloor$ mod n, where C is a positive integer (which you need to discover), w is an integer representation of a word, and n is the length of the table (that is, the number of suspect words). C must be as small as possible. Note that $\lfloor\ \rfloor$ is the floor function and that $\lfloor R \rfloor$ for some real number R is the largest integer that is less than or equal to R.

Assume the set of n suspect words is represented by the positive integers w_1, w_2, …, w_n. The problem is to find the smallest positive integer C such that $\lfloor C / w_i \rfloor$ mod $n \neq \lfloor C / w_j \rfloor$ mod n for all $1 \le i < j \le n$. You are to convert each input word to an integer by processing each letter in the word, working left to right. Consider 'a' to be 1, 'b' to be 2, …, and 'z' to be 26. Use 5 bits for each letter; before processing the next letter, shift the partially completed word's integer value left by 5 bits or multiply it by 32. Thus 'a' = 1, and 'bz' = $2 \times 32 + 26 = 90$.

C must be a multiple of at least one word's integer representation.

If $\lfloor C / w_i \rfloor$ mod $n = \lfloor C / w_j \rfloor$ mod n for some $i \neq j$ (that is, a hashing collision) then the next largest C that could possibly resolve the conflict is at least the minimum of $(\lfloor C / w_i \rfloor + 1) \times w_i$ and $(\lfloor C / w_j \rfloor + 1) \times w_j$. Since all conflicts must be resolved, it is advantageous to choose the largest candidate from among the conflicts as the next C to test. C will not always fit in a 32-bit integer.

Input

The input will contain multiple cases. For each case the input will consist of a single line containing between two and thirteen unique words, each containing only between one and five lowercase letters. The words are separated from each other by at least one blank.

Input for the last case will be followed by a line containing only an end of line character.

Output

For each case, display the case number (starting with 1) and the value of C that yields a perfect hashing function. Separate the output for consecutive cases with a blank line. Your output should appear similar to that shown in the sample below.

Sample Input	Output for the Sample Input

`bomb timer fuse radio`
`ax  knife  pick  lock gun`
This line contains only an end of line.
	`Case 1: 19029075`
	
	`Case 2: 817488`

Problem 9: Digital Editing

A display device is being constructed that will display certain five-digit values. Only certain values can be displayed, and the device can change only one digit at a time.

For example, suppose the allowed display values were 12345, 12346, 17345, 17346, 22346, and 26346. Is it possible to display each of these, given a choice as to which value is displayed first? The answer is yes, as illustrated below.

> Start with 26346.
> Change the first 6 to 2 to yield 22346.
> Now change the first 2 to 1 to yield 12346.
> Now change the remaining 2 to 7 to yield 17346.
> Then change the 6 to 5 to yield 17345.
> And finally, change the 7 to 2 to yield 12345.

Although all six values can be displayed in this case, it is not always the case that all values can be displayed. For example, consider this set of values: 59304, 58304, 8300, 48304, 19304, and 18304. The longest sequence that can be displayed has four values, as in 19304 → 59304 → 58304 → 58303.

Your job, given the set of values that can be displayed, is to determine the maximum number of values in a sequence that can be displayed, given that you can select the value with which the display begins, and that you can only change one digit at a time.

It is obvious that there are multiple such sequences. For example, reversing the order in which the values are displayed will yield another allowable sequence. It is not the particular sequence that is of interest here, but only the length of the sequence.

Input

The input will contain multiple cases. Each case begins with a line containing an integer *N* that specifies the number of distinct values in the set. *N* will be no larger than 10,000. This line is then followed by one or more lines containing a total of *N* unique integers, each in the range 0 to 99999, with one or more such integers per line. Multiple integers on a line will be separated by spaces.

Input for the last case will be followed by a line containing the integer 0.

Output

For each case, display the case number (starting with 1) and the number of unique values that can be displayed subject to the single-digit modification constraint. Separate output for consecutive cases with a blank line. Your output should appear similar to that shown in the sample below.

Sample Input

```
6
12345 12346 17345 17346 22346
26346
6
59304 58304 8300 48304 19304 58303
5
10057 57 10056 50056 58
0
```

Output for the Sample Input

```
Case 1. 6 values

Case 2. 4 values

Case 3. 5 values
```