01

Some of Easy UVA Problems

Set : 01
Problem Type : Ad Hoc
Problem Range : 100-999
Total Problem : 68

Xplosive
Nasif Ahmed
Email : nasif.002@gmail.com
Skype : nasif.02

INDEX OF PROBLEMS

34. 446 - Kibbles "n" Bits "n" Bits "n" Bits
35. 463 - Polynomial Factorization
36. 468 - Key to Success
37. 484 - The Department of Redundancy Department
38. 488 - Triangle Wave
39. 489 - Hangman Judge
40. 490 - Rotating Sentences
41. 495 - Fibonacci Freeze
42. 498 - Polly the Polynomial
43. 499 - What's The Frequency, Kenneth?
44. 530 - Binomial Showdown
45. 541 - Error Correction
46. 543 - Goldbach's Conjecture
47. 575 - Skew Binary
48. 579 - ClockHands
49. 583 - Prime Factors
50. 673 - Parentheses Balance
51. 686 - Goldbach's Conjecture (II)
52. 693 - Digital Racing Circuit
53. 694 - The Collatz Sequence
54. 729 - The Hamming Distance Problem
55. 834 - Continued Fractions
56. 863 - Process Scheduling
57. 894 - Juggling Trams
58. 896 - Board Game
59. 913 - Joana and the Odd Numbers
60. 917 - Euro 2004
61. 919 - Cutting Polyominoes
62. 951 - The Pieces of the Puzzle
63. 961 - Ambiguous or Incomplete Inductive Definitions
64. 964 - Custom Language
65. 977 - Old West Rumours
66. 973 - The Guessing Game
67. 984 - Finding Haplotypes

1. 102 - Ecological Bin Packing

Time limit: 3.000 seconds

Ecological Bin Packing

Background

Bin packing, or the placement of objects of certain weights into different bins subject to certain constraints, is an historically interesting problem. Some bin packing problems are NP-complete but are amenable to dynamic programming solutions or to approximately optimal heuristic solutions.

In this problem you will be solving a bin packing problem that deals with recycling glass.

The Problem

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass, and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.

The problem is to minimize the number of bottles that are moved. You may assume that the only problem is to minimize the number of movements between boxes.

For the purposes of this problem, each bin has infinite capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color. The total number of bottles will never exceed 2^31.

The Input

The input consists of a series of lines with each line containing 9 integers. The first three integers on a line represent the number of brown, green, and clear bottles (respectively) in bin number 1, the second three represent the number of brown, green and clear bottles (respectively) in bin number 2, and the last three integers represent the number of brown, green, and clear bottles (respectively) in bin number 3. For example, the line 10 15 20 30 12 8 15 8 31

indicates that there are 20 clear bottles in bin 1, 12 green bottles in bin 2, and 15 brown bottles in bin 3.

Integers on a line will be separated by one or more spaces. Your program should process all lines in the input file.

The Output

For each line of input there will be one line of output indicating what color bottles go in what bin to minimize the number of bottle movements. You should also print the minimum number of bottle movements.

The output should consist of a string of the three upper case characters 'G', 'B', 'C' (representing the colors green, brown, and clear) representing the color associated with each bin.

The first character of the string represents the color associated with the first bin, the second character of the string represents the color associated with the second bin, and the third character represents the color associated with the third bin.

The integer indicating the minimum number of bottle movements should follow the string.

If more than one order of brown, green, and clear bins yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

Sample Input

```
1 2 3 4 5 6 7 8 9
5 10 5 20 10 5 10 20 10
```

Sample Output

```
BCG 30
CBG 50
```

2. 103 - Stacking Boxes

Time limit: 3.000 seconds

Stacking Boxes

Background

Some concepts in Mathematics and Computer Science are simple in one or two dimensions but become more complex when extended to arbitrary dimensions. Consider solving differential equations in several dimensions and analyzing the topology of an *n*-dimensional hypercube. The former is much more complicated than its one dimensional relative while the latter bears a remarkable resemblance to its ``lower-class'' cousin.

The Problem

Consider an *n*-dimensional ``box'' given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box $4 \times 8 \times 9$ (length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of *n*-dimensional boxes. You are to determine the longest *nesting string* of boxes, that is a sequence of boxes $b_1, b_2, \ldots, b_k$ such that each box b_i nests in box b_{i+1} ($1 \le i < k$).

A box D = ($d_1, d_2, \ldots, d_n$) nests in a box E = ($e_1, e_2, \ldots, e_n$) if there is some rearrangement of the d_i such that when rearranged each dimension is less than the corresponding dimension in box E. This loosely corresponds to turning box D to see if it will fit in box E. However, since any rearrangement suffices, box D can be contorted, not just turned (see examples below).

For example, the box D = (2,6) nests in the box E = (7,3) since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E. The box D = (9,5,7,3) does NOT nest in the box E = (2,10,6,8) since no rearrangement of D results in a box that satisfies the nesting property, but F = (9,5,7,1) does nest in box E since F can be rearranged as (1,9,5,7) which nests in E.

Formally, we define nesting as follows: box D = ($d_1, d_2, \ldots, d_n$) *nests* in box E = ($e_1, e_2, \ldots, e_n$) if there is a permutation π of $1 \ldots n$ such that ($d_{\pi(1)}, d_{\pi(2)}, \ldots, d_{\pi(n)}$) ``fits'' in ($e_1, e_2, \ldots, e_n$) i.e., if $d_{\pi(i)} < e_i$ for all $1 \le i \le n$.

The Input

The input consists of a series of box sequences. Each box sequence begins with a line consisting of the the number of boxes *k* in the sequence followed by the dimensionality of the boxes, *n* (on the same line.)

This line is followed by *k* lines, one line per box with the *n* measurements of each box on one line separated by one or more spaces. The i^{th} line in the sequence ($1 \leq i \leq k$) gives the measurements for the i^{th} box.

There may be several box sequences in the input file. Your program should process all of them and determine, for each sequence, which of the *k* boxes determine the longest nesting string and the length of that nesting string (the number of boxes in the string).

In this problem the maximum dimensionality is 10 and the minimum dimensionality is 1. The maximum number of boxes in a sequence is 30.

The Output

For each box sequence in the input file, output the length of the longest nesting string on one line followed on the next line by a list of the boxes that comprise this string in order. The ``smallest" or ``innermost" box of the nesting string should be listed first, the next box (if there is one) should be listed second, etc.

The boxes should be numbered according to the order in which they appeared in the input file (first box is box 1, etc.).

If there is more than one longest nesting string then any one of them can be output.

Sample Input

```
5 2
3 7
8 10
5 2
9 11
21 18
8 6
5 2 20 1 30 10
23 15 7 9 11 3
40 50 34 24 14 4
9 10 11 12 13 14
31 4 18 8 27 17
44 32 13 19 41 19
1 2 3 4 5 6
80 37 47 18 21 9
```

Sample Output

```
5
3 1 2 4 5
4
7 2 5 6
```

3. 105 - The Skyline Problem

Time limit: 3.000 seconds

The Skyline Problem

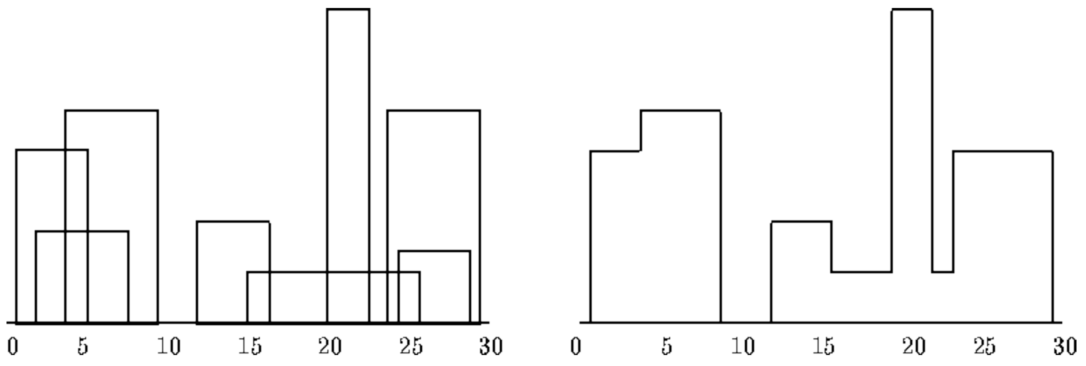
Background

With the advent of high speed graphics workstations, CAD (computer-aided design) and other areas (CAM, VLSI design) have made increasingly effective use of computers. One of the problems with drawing images is the elimination of hidden lines -- lines obscured by other parts of a drawing.

The Problem

You are to design a program to assist an architect in drawing the skyline of a city given the locations of the buildings in the city. To make the problem tractable, all buildings are rectangular in shape and they share a common bottom (the city they are built in is very flat). The city is also viewed as two-dimensional. A building is specified by an ordered triple (L_i, H_i, R_i) where L_i and R_i are left and right coordinates, respectively, of building *i* and H_i is the height of the building. In the diagram below buildings are shown on the left with triples (1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), (24,4,28)

the skyline, shown on the right, is represented by the sequence: (1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0)



The Input

The input is a sequence of building triples. All coordinates of buildings are positive integers less than 10,000 and there will be at least one and at most 5,000 buildings in the input file. Each building triple is on a line by itself in the input file. All integers in a triple are separated by one or more spaces. The triples will be sorted by L_i, the left *x*-coordinate of the building, so the building with the smallest left *x*-coordinate is first in the input file.

The Output

The output should consist of the vector that describes the skyline as shown in the example above. In the skyline vector $(v_1, v_2, v_3, \ldots, v_{n-2}, v_{n-1}, v_n)$, the v_i such that *i* is an even number represent a horizontal line (height). The v_i such that *i* is an odd number represent a vertical line (*x*-coordinate). The skyline vector should represent the ``path" taken, for example, by a bug starting at the minimum *x*-coordinate and traveling horizontally and vertically over all the lines that define the skyline. Thus the last entry in the skyline vector will be a 0. The coordinates must be separated by a blank space.

Sample Input

```
1 11 5
2 6 7
3 13 9
12 7 16
14 3 25
19 18 22
23 13 29
24 4 28
```

Sample Output

```
1 11 3 13 9 0 12 7 16 3 19 18 22 3 23 13 29 0
```

4. 108 - Maximum Sum

Time limit: 3.000 seconds

Maximum Sum

Background

A problem that is simple to solve in one dimension is often much more difficult to solve in more than one dimension. Consider satisfying a boolean expression in conjunctive normal form in which each conjunct consists of exactly 3 disjuncts. This problem (3-SAT) is NP-complete. The problem 2-SAT is solved quite efficiently, however. In contrast, some problems belong to the same complexity class regardless of the dimensionality of the problem.

The Problem

Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the*maximal sub-rectangle*. A sub-rectangle is any contiguous sub-array of size 1×1 or greater located within the whole array. As an example, the maximal sub-rectangle of the array:

$$\begin{array}{rrrr} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{array}$$

is in the lower-left-hand corner:

$$\begin{array}{rr} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{array}$$

and has the sum of 15.

Input and Output

The input consists of an $N \times N$ array of integers. The input begins with a single positive integer *N* on a line by itself indicating the size of the square two dimensional array. This is followed by N^2 integers separated by white-space (newlines and spaces). These N^2 integers make up the array in row-major order (i.e., all numbers on the first row, left-to-right, then all numbers on the second row, left-to-right, etc.). *N*may be as large as 100. The numbers in the array will be in the range [-127, 127].

The output is the sum of the maximal sub-rectangle.

Sample Input

```
4
0 -2 -7  0 9  2 -6  2
-4  1 -4  1 -1
8  0 -2
```

Sample Output

```
15
```

5. 109 - SCUD Busters

Time limit: 3.000 seconds

SCUD Busters

Background

Some problems are difficult to solve but have a simplification that is easy to solve. Rather than deal with the difficulties of constructing a model of the Earth (a somewhat oblate spheroid), consider a pre-Columbian flat world that is a 500 kilometer $\times$ 500 kilometer square.

In the model used in this problem, the flat world consists of several warring kingdoms. Though warlike, the people of the world are strict isolationists; each kingdom is surrounded by a high (but thin) wall designed to both protect the kingdom and to isolate it. To avoid fights for power, each kingdom has its own electric power plant.

When the urge to fight becomes too great, the people of a kingdom often launch missiles at other kingdoms. Each SCUD missile (Sanitary Cleansing Universal Destroyer) that lands within the walls of a kingdom destroys that kingdom's power plant (without loss of life).

The Problem

Given coordinate locations of several kingdoms (by specifying the locations of houses and the location of the power plant in a kingdom) and missile landings you are to write a program that determines the total area of all kingdoms that are without power after an exchange of missile fire.

In the simple world of this problem kingdoms do not overlap. Furthermore, the walls surrounding each kingdom are considered to be of zero thickness. The wall surrounding a kingdom is the minimal-perimeter wall that completely surrounds all the houses and the power station that comprise a kingdom; the area of a kingdom is the area enclosed by the minimal-perimeter thin wall.

There is exactly one power station per kingdom.

There may be empty space between kingdoms.

The Input

The input is a sequence of kingdom specifications followed by a sequence of missile landing locations.

A kingdom is specified by a number *N* ($3 \le N \le 100$) on a single line which indicates the number of sites in this kingdom. The next line contains the *x* and *y*coordinates of the power station, followed by *N*-1 lines of *x*, *y* pairs indicating the locations of homes served by this power station. A value of -1 for *N* indicates that there are no more kingdoms. There will be at least one kingdom in the data set.

Following the last kingdom specification will be the coordinates of one or more missile attacks, indicating the location of a missile landing. Each missile location is on a line by itself. You are to process missile attacks until you reach the end of the file.

Locations are specified in kilometers using coordinates on a 500 km by 500 km grid. All coordinates will be integers between 0 and 500 inclusive. Coordinates are specified as a pair of integers separated by white-space on a single line. The input file will consist of up to 20 kingdoms, followed by any number of missile attacks.

The Output

The output consists of a single number representing the total area of all kingdoms without electricity after all missile attacks have been processed. The number should be printed with (and correct to) two decimal places.

Sample Input

```
12
3 3
4 6
4 11
4 8
10 6
5 7
6 6
6 3
7 9
10 4
10 9
1 7
5
20 20
20 40
40 20
40 40
30 30
3
10 10
21 10
21 13
-1
5 5
20 12
```

Sample Output

```
70.50
```

A Hint

You may or may not find the following formula useful.

Given a polygon described by the vertices $v_0, v_1, \ldots, v_n$ such that $v_0 = v_n$, the signed area of the polygon is given by

$$a = \frac{1}{2}\sum_{i=1}^{n}(x_{i-1}y_i) - (x_i y_{i-1})$$

where the x, y coordinates of $v_i = (x_i, y_i)$; the edges of the polygon are from v_i to v_{i+1} for $i = 0 \ldots n-1$.

If the points describing the polygon are given in a counterclockwise direction, the value of *a* will be positive, and if the points of the polygon are listed in a clockwise direction, the value of *a* will be negative.

6. 111 - History Grading

Time limit: 3.000 seconds

History Grading

Background

Many problems in Computer Science involve maximizing some measure according to constraints.

Consider a history exam in which students are asked to put several historical events into chronological order. Students who order all the events correctly will receive full credit, but how should partial credit be awarded to students who incorrectly rank one or more of the historical events?

Some possibilities for partial credit include:

1. 1 point for each event whose rank matches its correct rank
2. 1 point for each event in the longest (not necessarily contiguous) sequence of events which are in the correct order relative to each other.

For example, if four events are correctly ordered 1 2 3 4 then the order 1 3 2 4 would receive a score of 2 using the first method (events 1 and 4 are correctly ranked) and a score of 3 using the second method (event sequences 1 2 4 and 1 3 4 are both in the correct order relative to each other).

In this problem you are asked to write a program to score such questions using the second method.

The Problem

Given the correct chronological order of *n* events $1, 2, \ldots, n$ as $c_1, c_2, \ldots c_n$ where $1 \leq c_i \leq n$ denotes the ranking of event *i* in the correct chronological order and a sequence of student responses $r_1, r_2, \ldots r_n$ where $1 \leq r_i \leq n$ denotes the chronological rank given by the student to event *i*; determine the length of the longest (not necessarily contiguous) sequence of events in the student responses that are in the correct chronological order relative to each other.

The Input

The first line of the input will consist of one integer *n* indicating the number of events with $2 \leq n \leq 20$. The second line will contain *n* integers, indicating the correct chronological order of *n* events. The remaining lines will each consist of *n* integers with each line representing a student's chronological ordering of the n events. All lines will contain *n* numbers in the range $[1 \ldots n]$, with each number appearing exactly once per line, and with each number separated from other numbers on the same line by one or more spaces.

The Output

For each student ranking of events your program should print the score for that ranking. There should be one line of output for each student ranking.

Sample Input 1

```
4
4 2 3 1
1 3 2 4
3 2 1 4
2 3 4 1
```

Sample Output 1

```
1
2
3
```

Sample Input 2

```
10
3 1 2 4 9 5 10 6 8 7
1 2 3 4 5 6 7 8 9 10
4 7 2 3 10 6 9 1 5 8
3 1 2 4 9 5 10 6 8 7
2 10 1 3 8 4 9 5 7 6
```

Sample Output 2

```
6
5
10
9
```

7. 113 - Power of Cryptography

Time limit: 3.000 seconds

Power of Cryptography

Background

Current work in cryptography involves (among other things) large prime numbers and computing powers of numbers modulo functions of these primes. Work in this area has resulted in the practical use of results from number theory and other branches of mathematics once considered to be of only theoretical interest.

This problem involves the efficient computation of integer roots of numbers.

The Problem

Given an integer $n \geq 1$ and an integer $p \geq 1$ you are to write a program that determines $\sqrt[n]{p}$, the positive n^{th} root of *p*. In this problem, given such integers *n* and*p*, *p* will always be of the form k^n for an integer *k* (this integer is what your program must find).

The Input

The input consists of a sequence of integer pairs *n* and *p* with each integer on a line by itself. For all such pairs $1 \leq n \leq 200$, $1 \leq p < 10^{101}$ and there exists an integer *k*, $1 \leq k \leq 10^9$ such that $k^n = p$.

The Output

For each integer pair *n* and *p* the value $\sqrt[n]{p}$ should be printed, i.e., the number *k* such that $k^n = p$.

Sample Input

```
2
16
3
27
7
4357186184021382204544
```

Sample Output

```
4
3
1234
```

8. 119 - Greedy Gift Givers

Time limit: 3.000 seconds

Greedy Gift Givers

The Problem

This problem involves determining, for a group of gift-giving friends, how much more each person gives than they receive (and vice versa for those that view gift-giving with cynicism).

In this problem each person sets aside some money for gift-giving and divides this money evenly among all those to whom gifts are given.

However, in any group of friends, some people are more giving than others (or at least may have more acquaintances) and some people have more money than others.

Given a group of friends, the money each person in the group spends on gifts, and a (sub)list of friends to whom each person gives gifts; you are to write a program that determines how much more (or less) each person in the group gives than they receive.

The Input

The input is a sequence of gift-giving groups. A group consists of several lines:

- the number of people in the group,
- a list of the names of each person in the group,
- a line for each person in the group consisting of the name of the person, the amount of money spent on gifts, the number of people to whom gifts are given, and the names of those to whom gifts are given.

All names are lower-case letters, there are no more than 10 people in a group, and no name is more than 12 characters in length. Money is a non-negative integer less than 2000.

The input consists of one or more groups and is terminated by end-of-file.

The Output

For each group of gift-givers, the name of each person in the group should be printed on a line followed by the net gain (or loss) received (or spent) by the person. Names in a group should be printed in the same order in which they first appear in the input.

The output for each group should be separated from other groups by a blank line. All gifts are integers. Each person gives the same integer amount of money to each friend to whom any money is given, and gives as much as possible. Any money not given is kept and is part of a person's ``net worth" printed in the output.

Sample Input

```
5
dave laura owen vick amr
dave 200 3 laura owen vick
owen 500 1 dave
amr 150 2 vick owen
laura 0 2 amr vick
vick 0 0
3
liz steve dave
liz 30 1 steve
steve 55 2 liz dave
dave 0 2 steve liz
```

Sample Output

```
dave 302
laura 66
owen -359
vick 141
amr -150

liz -3
steve -24
dave 27
```

9. 120 - Stacks of Flapjacks

Time limit: 3.000 seconds

Stacks of Flapjacks

Background

Stacks and Queues are often considered the bread and butter of data structures and find use in architecture, parsing, operating systems, and discrete event simulation. Stacks are also important in the theory of formal languages.

This problem involves both butter and sustenance in the form of pancakes rather than bread in addition to a finicky server who flips pancakes according to a unique, but complete set of rules.

The Problem

Given a stack of pancakes, you are to write a program that indicates how the stack can be sorted so that the largest pancake is on the bottom and the smallest pancake is on the top. The size of a pancake is given by the pancake's diameter. All pancakes in a stack have different diameters.

Sorting a stack is done by a sequence of pancake ``flips''. A flip consists of inserting a spatula between two pancakes in a stack and flipping (reversing) the pancakes on the spatula (reversing the sub-stack). A flip is specified by giving the position of the pancake on the bottom of the sub-stack to be flipped (relative to the whole stack). The pancake on the bottom of the whole stack has position 1 and the pancake on the top of a stack of *n* pancakes has position *n*.

A stack is specified by giving the diameter of each pancake in the stack in the order in which the pancakes appear.

For example, consider the three stacks of pancakes below (in which pancake 8 is the top-most pancake of the left stack):

```
8           7           2
4           6           5
6           4           8
7           8           4
5           5           6
2           2           7
```

The stack on the left can be transformed to the stack in the middle via *flip(3)*. The middle stack can be transformed into the right stack via the command *flip(1)*.

The Input

The input consists of a sequence of stacks of pancakes. Each stack will consist of between 1 and 30 pancakes and each pancake will have an integer diameter between 1 and 100. The input is terminated by end-of-file. Each stack is given as a single line of input with the top pancake on a stack appearing first on a line, the bottom pancake appearing last, and all pancakes separated by a space.

The Output

For each stack of pancakes, the output should echo the original stack on one line, followed by some sequence of flips that results in the stack of pancakes being sorted so that the largest diameter pancake is on the bottom and the smallest on top. For each stack the sequence of flips should be terminated by a 0 (indicating no more flips necessary). Once a stack is sorted, no more flips should be made.

Sample Input

```
1 2 3 4 5
5 4 3 2 1
5 1 2 3 4
```

Sample Output

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

10. 130 - Roman Roulette

Time limit: 3.000 seconds

Roman Roulette

The historian Flavius Josephus relates how, in the Romano-Jewish conflict of 67 A.D., the Romans took the town of Jotapata which he was commanding. Escaping, Jospehus found himself trapped in a cave with 40 companions. The Romans discovered his whereabouts and invited him to surrender, but his companions refused to allow him to do so. He therefore suggested that they kill each other, one by one, the order to be decided by lot. Tradition has it that the means for effecting the lot was to stand in a circle, and, beginning at some point, count round, every third person being killed in turn. The sole survivor of this process was Josephus, who then surrendered to the Romans. Which begs the question: had Josephus previously practised quietly with 41 stones in a dark corner, or had he calculated mathematically that he should adopt the 31st position in order to survive?

Having read an account of this gruesome event you become obsessed with the fear that you will find yourself in a similar situation at some time in the future. In order to prepare yourself for such an eventuality you decide to write a program to run on your hand-held PC which will determine the position that the counting process should start in order to ensure that you will be the sole survivor.

In particular, your program should be able to handle the following variation of the processes described by Josephus. $n > 0$ people are initially arranged in a circle, facing inwards, and numbered from 1 to n. The numbering from 1 to n proceeds consecutively in a clockwise direction. Your allocated number is 1. Starting with person number i, counting starts in a clockwise direction, until we get to person number$k$ ($k > 0$), who is promptly killed. We then proceed to count a further k people in a clockwise direction, starting with the person immediately to the left of the victim. The person number k so selected has the job of burying the victim, and then returning to the position in the circle that the victim had previously occupied. Counting then proceeds from the person to his immediate left, with the kth person being killed, and so on, until only one person remains.

For example, when $n = 5$, and $k = 2$, and $i = 1$, the order of execution is 2, 5, 3, and 1. The survivor is 4.

Input and Output

Your program must read input lines containing values for n and k (in that order), and for each input line output the number of the person with which the counting should begin in order to ensure that you are the sole survivor. For example, in the above case the safe starting position is 3. Input will be terminated by a line containing values of 0 for n and k.

Your program may assume a maximum of 100 people taking part in this event.

Sample Input

```
1 1
1 5
0 0
```

Sample Output

```
1
1
```

11. 133 - The Dole Queue

Time limit: 3.000 seconds

The Dole Queue

In a serious attempt to downsize (reduce) the dole queue, The New National Green Labour Rhinoceros Party has decided on the following strategy. Every day all dole applicants will be placed in a large circle, facing inwards. Someone is arbitrarily chosen as number 1, and the rest are numbered counter-clockwise up to N (who will be standing on 1's left). Starting from 1 and moving counter-clockwise, one labour official counts off k applicants, while another official starts from N and moves clockwise, counting m applicants. The two who are chosen are then sent off for retraining; if both officials pick the same person she (he) is sent off to become a politician. Each official then starts counting again at the next available person and the process continues until no-one is left. Note that the two victims (sorry, trainees) leave the ring simultaneously, so it is possible for one official to count a person already selected by the other official.

Input

Write a program that will successively read in (in that order) the three numbers (N, k and m; k, m > 0, 0 < N < 20) and determine the order in which the applicants are sent off for retraining. Each set of three numbers will be on a separate line and the end of data will be signalled by three zeroes (0 0 0).

Output

For each triplet, output a single line of numbers specifying the order in which people are chosen. Each number should be in a field of 3 characters. For pairs of numbers list the person chosen by the counter-clockwise official first. Separate successive pairs (or singletons) by commas (but there should not be a trailing comma).

Sample input

```
10 4 3
0 0 0
```

Sample output

```
△△4△△8,△△9△△5,△△3△△1,△△2△△6,△10,△△7
```

where △ represents a space.

12. 146 - ID Codes

Time limit: 3.000 seconds

ID Codes

It is 2084 and the year of Big Brother has finally arrived, albeit a century late. In order to exercise greater control over its citizens and thereby to counter a chronic breakdown in law and order, the Government decides on a radical measure--all citizens are to have a tiny microcomputer surgically implanted in their left wrists. This computer will contains all sorts of personal information as well as a transmitter which will allow people's movements to be logged and monitored by a central computer. (A desirable side effect of this process is that it will shorten the dole queue for plastic surgeons.)

An essential component of each computer will be a unique identification code, consisting of up to 50 characters drawn from the 26 lower case letters. The set of characters for any given code is chosen somewhat haphazardly. The complicated way in which the code is imprinted into the chip makes it much easier for the manufacturer to produce codes which are rearrangements of other codes than to produce new codes with a different selection of letters. Thus, once a set of letters has been chosen all possible codes derivable from it are used before changing the set.

For example, suppose it is decided that a code will contain exactly 3 occurrences of `a', 2 of `b' and 1 of `c', then three of the allowable 60 codes under these conditions are:

```
abaabc
abaacb
ababac
```

These three codes are listed from top to bottom in alphabetic order. Among all codes generated with this set of characters, these codes appear consecutively in this order.

Write a program to assist in the issuing of these identification codes. Your program will accept a sequence of no more than 50 lower case letters (which may contain repeated characters) and print the successor code if one exists or the message `No Successor' if the given code is the last in the sequence for that set of characters.

Input and Output

Input will consist of a series of lines each containing a string representing a code. The entire file will be terminated by a line consisting of a single #.

Output will consist of one line for each code read containing the successor code or the words `No Successor'.

Sample input

```
abaacb
cbbaa
#
```

Sample output

```
ababac
No Successor
```

13. 151 - Power Crisis

Time limit: 3.000 seconds

Power Crisis

During the power crisis in New Zealand this winter (caused by a shortage of rain and hence low levels in the hydro dams), a contingency scheme was developed to turn off the power to areas of the country in a systematic, totally fair, manner. The country was divided up into *N* regions (Auckland was region number 1, and Wellington number 13). A number, *m*, would be picked `at random', and the power would first be turned off in region 1 (clearly the fairest starting point) and then in every m'th region after that, wrapping around to 1 after *N*, and ignoring regions already turned off. For example, if $N = 17$ and $m = 5$, power would be turned off to the regions in the order:1,6,11,16,5,12,2,9,17,10,4,15,14,3,8,13,7.

The problem is that it is clearly fairest to turn off Wellington last (after all, that is where the Electricity headquarters are), so for a given *N*, the `random' number *m* needs to be carefully chosen so that region 13 is the last region selected.

Write a program that will read in the number of regions and then determine the smallest number *m* that will ensure that Wellington (region 13) can function while the rest of the country is blacked out.

Input and Output

Input will consist of a series of lines, each line containing the number of regions (*N*) with $13 \le N < 100$. The file will be terminated by a line consisting of a single 0.

Output will consist of a series of lines, one for each line of the input. Each line will consist of the number *m* according to the above scheme.

Sample input

```
17
0
```

Sample output

```
7
```

14. 190 - Circle Through Three Points

Time limit: 3.000 seconds

Circle Through Three Points

Your team is to write a program that, given the Cartesian coordinates of three points on a plane, will find the equation of the circle through them all. The three points will not be on a straight line.

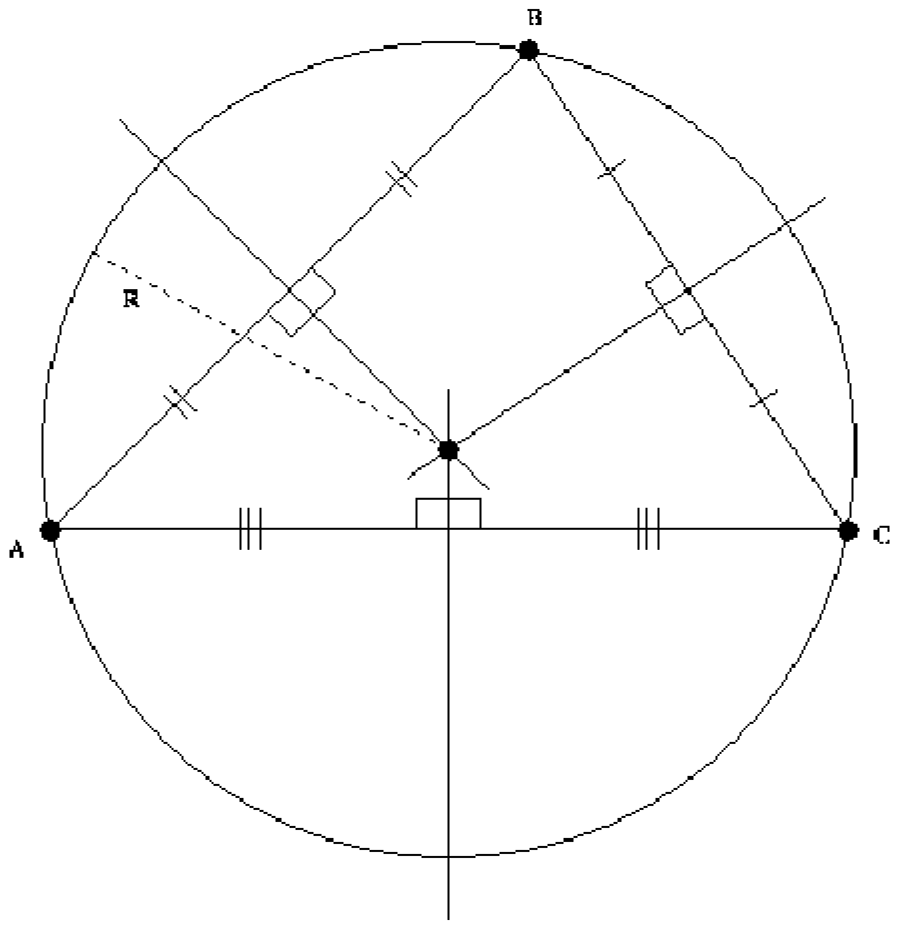
The solution is to be printed as an equation of the form

$$(x-h)^2+(y-k)^2=r^2 \qquad (1)$$

and an equation of the form

$$x^2+y^2+cx+dy+e=0 \qquad (2)$$

Each line of input to your program will contain the *x* and *y* coordinates of three points, in the order A_x, A_y, B_x, B_y, C_x, C_y. These coordinates will be real numbers separated from each other by one or more spaces.



Your program must print the required equations on two lines using the format given in the sample below. Your computed values for *h*, *k*, *r*, *c*, *d*, and *e* in Equations 1 and 2 above are to be printed with three digits after the decimal point. Plus and minus signs in the equations should be changed as needed to avoid multiple signs before a number. Plus, minus, and equal signs must be separated from the adjacent characters by a single space on each side. No other spaces are to appear in the equations. Print a single blank line after each equation pair.

Sample input

```
7.0 -5.0 -1.0 1.0 0.0 -6.0
1.0 7.0 8.0 6.0 7.0 -2.0
```

Sample output

```
(x - 3.000)^2 + (y + 2.000)^2 = 5.000^2
x^2 + y^2 - 6.000x + 4.000y - 12.000 = 0

(x - 3.921)^2 + (y - 2.447)^2 = 5.409^2
x^2 + y^2 - 7.842x - 4.895y - 7.895 = 0
```

15. 219 - Department of Redundancy Department

Time limit: 3.000 seconds

Department of Redundancy Department

When designing tables for a relational database, a functional dependency (FD) is used to express the relationship between the different fields. A functional dependency is concerned with the relationship of values of one set of fields to those of another set of fields.

The notation `X->Y` is used to denote that when supplied values to the field(s) in set `X`, the assigned value for each field in set `Y` can be determined. For example, if a database table is to contain fields for the *social security number* (`S`), *name* (`N`), *address* (`A`), and *phone* (`P`) and each person has been assigned a unique value for `S`, the `S` field functionally determines the `N`, `A` and `P` fields. This is written as `S->NAP`.

Develop a program that will identify each redundant FD in each input group of FDs. An FD is redundant if it can be derived using other FDs in the group.

For example, if the group contains the FDs `A->B`, `B->C`, and `A->C`, then the third FD is redundant since the field set `C` can be derived using the first two. (The `A` fields determine values for the `B` fields, which in turn determine values for the fields in `C`.) In the group `A->B`, `B->C`, `C->A`, `A->C`, `C->B`, and `B->A`, all the FDs are redundant.

Input

The input file contains an arbitrary number of groups of FDs. Each group is preceded by a line containing an integer no larger than 100 specifying the number of FDs in that group. A group with zero FDs indicates the end of the input.

Each FD in the group appears on a separate line containing two non-empty lists of field names separated by the characters `-` and `>`. The lists of field names contain only uppercase alphabetic characters. Functional dependency lines contain no blanks or tabs. There are no trivially redundant FDs (for example, `A->A`).

For identification purposes, groups are numbered sequentially, starting with 1; the FDs are also numbered sequentially, starting with 1 in each group.

Output

For each group, in order, your program must identify the group, each redundant FD in the group, and a sequence of the other FDs in the group which were used to determine the indicated FD is redundant. If more than one sequence of FDs can be used to show another FD is redundant, any such sequence is acceptable, even if it is not the shortest proof sequence. Each FD in an acceptable proof sequence must, however, be necessary.

If a group of FDs contains no redundancy, display `No redundant FDs.`

There should be a blank line after every test case

Sample Input

```
3
A->BD
BD->C
A->C
6
P->RST
VRT->SQP
PS->T
Q->TR
QS->P
SR->V
5
A->B
A->C
B->D
C->D
A->D
3
A->B
B->C
A->D
0
```

Sample Output

```
Set number 1
     FD 3 is redundant using FDs: 1 2

Set number 2
     FD 3 is redundant using FDs: 1
     FD 5 is redundant using FDs: 4 6 2

Set number 3
     FD 5 is redundant using FDs: 1 3
          --OR--
     FD 5 is redundant using FDs: 2 4

Set number 4
     No redundant FDs.
```

16.236 - VTAS - Vessel Traffic Advisory Service

Time limit: 3.000 seconds

VTAS - Vessel Traffic Advisory Service

In order to promote safety and efficient use of port facilities, the Association of Coastal Merchants (ACM) has developed a concept for a Vessel Traffic Advisory Service (VTAS) that will provide traffic advisories for vessels transiting participating ports.

The concept is built on a computer program that maintains information about the traffic patterns and reported movements of vessels within the port over multiple days. For each port, the traffic lanes are defined between waypoints. The traffic lanes have been designated as directional to provide traffic separation and flow controls. Each port is represented by a square matrix containing the distances (in nautical miles) along each valid traffic lane. The distances are defined from each row waypoint to each column waypoint. A distance of 0 indicates that no valid traffic lane exists between the two waypoints.

Vessel traffic enters the port at a waypoint and transits the traffic lanes. A vessel may begin its transit at any of the waypoints and must follow a valid connected route via the valid traffic lanes. A vessel may end its transit at any valid waypoint.

The service provided by the VTAS to transiting vessels includes:

- Projection of arrival times at waypoints
- Notification of invalid routes
- Projected encounters with other vessels on each leg of the transit. An ``encounter" occurs when two vessels are between common waypoints (either traffic lane) at a common time
- Warning of close passing with another vessel in the vicinity of a waypoint (within 3 minutes of projected waypoint arrival)

Reported times will be rounded to the nearest whole minute. Time is maintained based on a 24 hour clock (i.e. 9 am is 0900, 9 PM is 2100, midnight is 0000). Speed is measured in knots which is equal to 1 nautical mile per hour.

Input

The input file for the computer program includes several datasets, each containing a Port Specification to provide the description of the traffic patterns within the port and a Traffic List which contains the sequence of vessels entering the port and their intended tracks. The end of each dataset is indicated by a Vessel Name beginning with an ``*".

Port Specification:	Number of Waypoints in Port (an integer *N*)
	Waypoint ID List (*N* single-character designators)
	Waypoint Connection Matrix (*N* rows of *N* real values specifying the distances between waypoints in nautical miles)
Traffic List:	Vessel Name (alphabetic characters)
	Time at first waypoint (on 24-hour clock) & Planned Transit Speed (in knots)
	Planned Route (ordered list of waypoints)

Output

The output for each dataset shall provide for each vessel as it enters the port a listing indicating the arrival of the vessel and its planned speed followed by a table containing the waypoints in its route and projected arrival at each waypoint. Following this table will be appropriate messages indicating:

- Notification of Invalid Routes
- Projected Encounters on each leg
- Warning of close passing at waypoints

All times are to be printed as four-digit integers with leading zeros when necessary. Print a blank line after each dataset.

Assumptions & Limitations:

1. Vessel names are at most 20 characters long.

2. There are at most 20 waypoints in a port and at most 20 waypoints in any route.

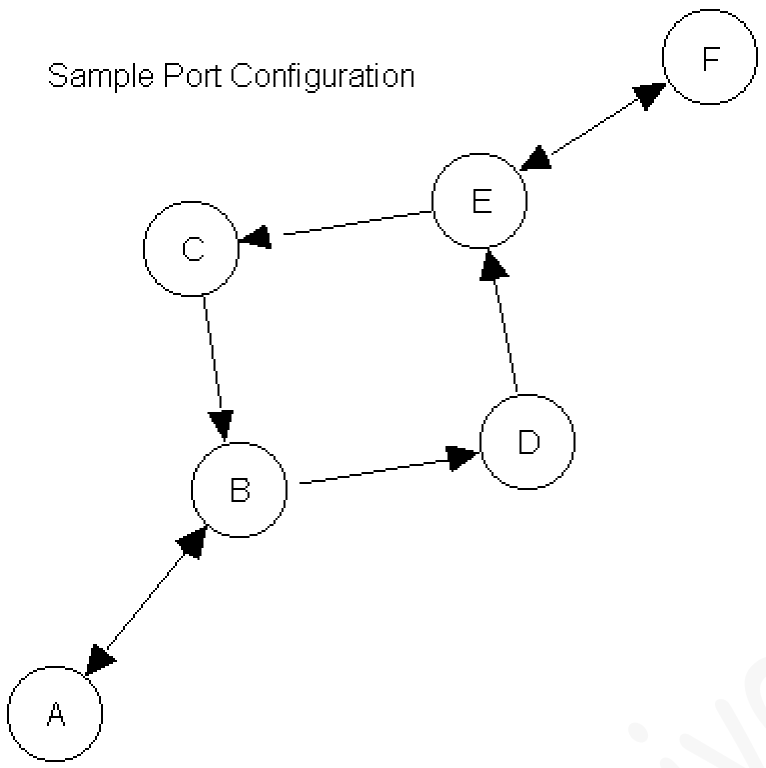
3. There will be at most 20 vessels in port at any time.

4. A vessel will complete its transit in at most 12 hours.

5. No more than 24 hours will elapse between vessel entries.

Sample Input

```
6
ABCDEF
0              3              0              0              0              0
3              0              0              2              0              0
0              3              0              0              0              0
0              0              0              0              3              0
0              0              2              0              0              4
0              0              0              0              4              0
Tug
2330                                                                        12
ABDEF
WhiteSailboat
2345                                                                         6
ECBDE
TugWBarge
2355                                                                         5
DECBA
PowerCruiser
0                                                                           15
FECBA
LiberianFreighter
7                                                                           18
ABDXF
ChineseJunk
45                                                                           8
ACEF
*****
```



Sample Output

```
Tug entering system at 2330 with a planned speed of 12.0 knots
          Waypoint:    A    B    D    E    F
          Arrival:   2330 2345 2355 0010 0030

WhiteSailboat entering system at 2345 with a planned speed of 6.0 knots
          Waypoint:    E    C    B    D    E
          Arrival:   2345 0005 0035 0055 0125

TugWBarge entering system at 2355 with a planned speed of 5.0 knots
          Waypoint:    D    E    C    B    A
          Arrival:   2355 0031 0055 0131 0207
Projected encounter with Tug on leg between Waypoints D & E
** Warning ** Close passing with Tug at Waypoint D

PowerCruiser entering system at 0000 with a planned speed of 15.0 knots
          Waypoint:    F    E    C    B    A
          Arrival:   0000 0016 0024 0036 0048
Projected encounter with Tug on leg between Waypoints F & E
Projected encounter with WhiteSailboat on leg between Waypoints C & B
** Warning ** Close passing with WhiteSailboat at Waypoint B

LiberianFreighter entering system at 0007 with a planned speed of 18.0 knots
**> Invalid Route Plan for Vessel: LiberianFreighter

ChineseJunk entering system at 0045 with a planned speed of 8.0 knots
**> Invalid Route Plan for Vessel: ChineseJunk
```

17. 256 - Quirksome Squares

Time limit: 3.000 seconds

Quirksome Squares

The number 3025 has a remarkable quirk: if you split its decimal representation in two strings of equal length (30 and 25) and square the sum of the numbers so obtained, you obtain the original number:

$$(30 + 25)^2 = 3025$$

The problem is to determine all numbers with this property having a given even number of digits.

For example, 4-digit numbers run from 0000 to 9999. Note that leading zeroes should be taken into account. This means that 0001 which is equal to $(00 + 01)^2$ is a quirksome number of 4 digits. The number of digits may be 2,4,6 or 8. Although maxint is only 32767 and numbers of eight digits are asked for, a well-versed programmer can keep his numbers in the range of the integers. However efficiency should be given a thought.

Input

The input of your program is a textflle containing numbers of digits (taken from 2,4,6,8), each number on a line of its own.

Output

The output is a textfile consisting of lines containing the quirksome numbers (ordered according to the input numbers and for each input number in increasing order).

Warning: Please note that the number of digits in the output is equal to the number in the corresponding input line : leading zeroes may not be suppressed.

Sample Input

```
2
2
```

Sample Output

```
00
01
81
00
01
81
```

18. 264 - Count on Cantor

Time limit: 3.000 seconds

Count on Cantor

One of the famous proofs of modern mathematics is Georg Cantor's demonstration that the set of rational numbers is enumerable. The proof works by using an explicit enumeration of rational numbers as shown in the diagram below.

```
1/1  1/2  1/3  1/4  1/5  ...
2/1  2/2  2/3  2/4
3/1  3/2  3/3
4/1  4/2
5/1
```

In the above diagram, the first term is 1/1, the second term is 1/2, the third term is 2/1, the fourth term is 3/1, the fifth term is 2/2, and so on.

Input and Output

You are to write a program that will read a list of numbers in the range from 1 to 10^7 and will print for each number the corresponding term in Cantor's enumeration as given below. No blank line should appear after the last number.

The input list contains a single number per line and will be terminated by end-of-file.

Sample input

```
3
14
7
```

Sample output

```
TERM 3 IS 2/1
TERM 14 IS 2/4
TERM 7 IS 1/4
```

19. 305 - Joseph

Time limit: 3.000 seconds

Joseph

The Joseph's problem is notoriously known. For those who are not familiar with the original problem: from among *n* people, numbered 1, 2, ..., *n*, standing in circle every*m*th is going to be executed and only the life of the last remaining person will be saved. Joseph was smart enough to choose the position of the last remaining person, thus saving his life to give us the message about the incident. For example when $n = 6$ and$m = 5$ then the people will be executed in the order 5, 4, 6, 2, 3 and 1 will be saved.

Suppose that there are *k* good guys and *k* bad guys. In the circle the first *k* are good guys and the last *k* bad guys. You have to determine such minimal *m* that all the bad guys will be executed before the first good guy.

Input

The input file consists of separate lines containing *k*. The last line in the input file contains 0. You can suppose that $0 < k < 14$.

Output

The output file will consist of separate lines containing *m* corresponding to *k* in the input file.

Sample Input

```
3
4
0
```

Sample Output

```
5
30
```

20. 324 - Factorial Frequencies

Time limit: 3.000 seconds

Factorial Frequencies

In an attempt to bolster her sagging palm-reading business, Madam Phoenix has decided to offer several numerological treats to her customers. She has been able to convince them that the frequency of occurrence of the digits in the decimal representation of factorials bear witness to their futures. Unlike palm-reading, however, she can't just conjure up these frequencies, so she has employed you to determine these values.

Recall that the definition of *n*! (that is, *n* factorial) is just $1 \times 2 \times 3 \times \ldots \times n$. As she expects to use either the day of the week, the day of the month, or the day of the year as the value of *n*, you must be able to determine the number of occurrences of each decimal digit in numbers as large as 366 factorial (366!), which has 781 digits.

Input and Output

The input data for the program is simply a list of integers for which the digit counts are desired. All of these input values will be less than or equal to 366 and greater than 0, except for the last integer, which will be zero. Don't bother to process this zero value; just stop your program at that point. The output format isn't too critical, but you should make your program produce results that look similar to those shown below.

Madam Phoenix will be forever (or longer) in your debt; she might even give you a trip if you do your job well!

Sample Input

```
3
8
100
0
```

Sample Output

```
3! --
   (0)    0    (1)    0    (2)    0    (3)    0    (4)    0
   (5)    0    (6)    1    (7)    0    (8)    0    (9)    0
8! --
   (0)    2    (1)    0    (2)    1    (3)    1    (4)    1
   (5)    0    (6)    0    (7)    0    (8)    0    (9)    0
100! --
   (0)   30    (1)   15    (2)   19    (3)   10    (4)   10
   (5)   14    (6)   19    (7)    7    (8)   14    (9)   20
```

21. 344 - Roman Digititis

Time limit: 3.000 seconds

Roman Digititis

Many persons are familiar with the Roman numerals for relatively small numbers. The symbols ``i", ``v", ``x", ``l", and ``c" represent the decimal values 1, 5, 10, 50, and 100 respectively. To represent other values, these symbols, and multiples where necessary, are concatenated, with the smaller-valued symbols written further to the right. For example, the number 3 is represented as ``iii", and the value 73 is represented as ``lxxiii". The exceptions to this rule occur for numbers having units values of 4 or 9, and for tens values of 40 or 90. For these cases, the Roman numeral representations are ``iv" (4), ``ix" (9), ``xl" (40), and ``xc" (90). So the Roman numeral representations for 24, 39, 44, 49, and 94 are ``xxiv", ``xxxix", ``xliv", ``xlix", and ``xciv", respectively.

The preface of many books has pages numbered with Roman numerals, starting with ``i" for the first page of the preface, and continuing in sequence. Assume books with pages having 100 or fewer pages of preface. How many ``i", ``v", ``x", ``l", and ``c" characters are required to number the pages in the preface? For example, in a five page preface we�ll use the Roman numerals ``i", ``ii", ``iii", ``iv", and ``v", meaning we need 7 ``i" characters and 2 ``v" characters.

Input

The input will consist of a sequence of integers in the range 1 to 100, terminated by a zero. For each such integer, except the final zero, determine the number of different types of characters needed to number the prefix pages with Roman numerals.

Output

For each integer in the input, write one line containing the input integer and the number of characters of each type required. The examples shown below illustrate an acceptable format.

Sample Input

```
1
2
20
99
0
```

22. 353 - Pesky Palindromes

Time limit: 3.000 seconds

Pesky Palindromes

A palindrome is a sequence of one or more characters that reads the same from the left as it does from the right. For example, Z, TOT and MADAM are palindromes, but ADAM is not.

Your job, should you choose to accept it, is to write a program that reads a sequence of strings and for each string determines the number of UNIQUE palindromes that are substrings.

Input and Output

The input file consists of a number of strings (one per line), of at most 80 characters each, starting in column 1.

For each non-empty input line, the output consists of one line containing the message:

`The string '`*input string*`' contains `*nnnn*` palindromes.`

where *input string* is replaced by the actual input string and *nnnn* is replaced by the number of UNIQUE palindromes that are substrings.

For input string ADAM, the UNIQUE palindromes are A, D, M and ADA so the correct output would be

```
The string 'ADAM' contains 4 palindromes.
```

Sample input

```
boy
adam
madam
tot
```

Sample output

```
The string 'boy' contains 3 palindromes.
The string 'adam' contains 4 palindromes.
The string 'madam' contains 5 palindromes.
The string 'tot' contains 3 palindromes.
```

Note

The 3 unique palindromes in 'boy' are 'b', 'o' and 'y'.
The 4 unique palindromes in 'adam' are 'a', 'd', 'm', and 'ada'.
The 5 unique palindromes in 'madam' are 'm', 'a', 'd', 'ada', and 'madam'.
The 3 unique palindromes in 'tot' are 't', 'o' and 'tot'.

Sample Output

```
1: 1 i, 0 v, 0 x, 0 l, 0 c
2: 3 i, 0 v, 0 x, 0 l, 0 c
20: 28 i, 10 v, 14 x, 0 l, 0 c
99: 140 i, 50 v, 150 x, 50 l, 10 c
```

23. 369 - Combinations

Time limit: 3.000 seconds

Combinations

Computing the exact number of ways that N things can be taken M at a time can be a great challenge when N and/or M become very large. Challenges are the stuff of contests. Therefore, you are to make just such a computation given the following:

GIVEN:

$$5 \le N \le 100, \quad \text{and} \quad 5 \le M \le 100, \quad \text{and} \quad M \le N$$

Compute the **EXACT** value of:

$$C = \frac{N!}{(N-M)! \times M!}$$

You may assume that the final value of C will fit in a 32-bit Pascal LongInt or a C long.

For the record, the exact value of 100! is:

93,326,215,443,944,152,681,699,238,856,266,700,490,715,968,264,381,621,
468,592,963,895,217,599,993,229,915,608,941,463,976,156,518,286,253,
697,920,827,223,758,251,185,210,916,864,000,000,000,000,000,000,000,000

Input and Output

The input to this program will be one or more lines each containing zero or more leading spaces, a value for N, one or more spaces, and a value for M. The last line of the input file will contain a dummy N, M pair with both values equal to zero. Your program should terminate when this line is read.

The output from this program should be in the form:

N `things taken` M `at a time is` C `exactly.`

Sample Input

```
100  6
 20  5
 18  6
  0  0
```

Sample Output

```
100 things taken 6 at a time is 1192052400 exactly.
20 things taken 5 at a time is 15504 exactly.
18 things taken 6 at a time is 18564 exactly.
```

24. 382 - Perfection

Time limit: 3.000 seconds

Perfection

From the article Number Theory in the 1994 Microsoft Encarta: ``If *a*, *b*, *c* are integers such that *a* = *bc*, *a* is called a multiple of *b* or of *c*, and *b* or *c* is called a divisor or factor of *a*. If *c* is not ± 1, *b* is called a proper divisor of *a*. Even integers, which include 0, are multiples of 2, for example, -4, 0, 2, 10; an odd integer is an integer that is not even, for example, -5, 1, 3, 9. A perfect number is a positive integer that is equal to the sum of all its positive, proper divisors; for example, 6, which equals 1 + 2 + 3, and 28, which equals 1 + 2 + 4 + 7 + 14, are perfect numbers. A positive number that is not perfect is imperfect and is deficient or abundant according to whether the sum of its positive, proper divisors is smaller or larger than the number itself. Thus, 9, with proper divisors 1, 3, is deficient; 12, with proper divisors 1, 2, 3, 4, 6, is abundant."

Problem Statement

Given a number, determine if it is perfect, abundant, or deficient.

Input

A list of *N* positive integers (none greater than 60,000), with $1 < N < 100$. A 0 will mark the end of the list.

Output

The first line of output should read `PERFECTION OUTPUT`. The next *N* lines of output should list for each input integer whether it is perfect, deficient, or abundant, as shown in the example below. Format counts: the echoed integers should be right justified within the first 5 spaces of the output line, followed by two blank spaces, followed by the description of the integer. The final line of output should read `END OF OUTPUT`.

Sample Input

```
15 28 6 56 60000 22 496 0
```

Sample Output

```
PERFECTION OUTPUT
   15  DEFICIENT
   28  PERFECT
    6  PERFECT
   56  ABUNDANT
60000  ABUNDANT
   22  DEFICIENT
  496  PERFECT
END OF OUTPUT
```

25. 401 - Palindromes

Time limit: 3.000 seconds

Palindromes

A regular palindrome is a string of numbers or letters that is the same forward as backward. For example, the string `"ABCDEDCBA"` is a palindrome because it is the same when the string is read from left to right as when the string is read from right to left.

A mirrored string is a string for which when each of the elements of the string is changed to its reverse (if it has a reverse) and the string is read backwards the result is the same as the original string. For example, the string `"3AIAE"` is a mirrored string because `"A"` and `"I"` are their own reverses, and `"3"` and `"E"` are each others' reverses.

A mirrored palindrome is a string that meets the criteria of a regular palindrome and the criteria of a mirrored string. The string `"ATOYOTA"` is a mirrored palindrome because if the string is read backwards, the string is the same as the original and because if each of the characters is replaced by its reverse and the result is read backwards, the result is the same as the original string. Of course, `"A"`, `"T"`, `"O"`, and `"Y"` are all their own reverses.
A list of all valid characters and their reverses is as follows.

Character	Reverse	Character	Reverse	Character	Reverse
A	A	M	M	Y	Y
B		N		Z	5
C		O	O	1	1
D		P		2	S
E	3	Q		3	E
F		R		4	
G		S	2	5	Z
H	H	T	T	6	
I	I	U	U	7	
J	L	V	V	8	8
K		W	W	9	
L	J	X	X		

Note that O (zero) and 0 (the letter) are considered the same character and therefore**ONLY** the letter "0" is a valid character.

Input

Input consists of strings (one per line) each of which will consist of one to twenty valid characters. There will be no invalid characters in any of the strings. Your program should read to the end of file.

Output

For each input string, you should print the string starting in column 1 immediately followed by exactly one of the following strings.

STRING	CRITERIA
`" -- is not a palindrome."`	if the string is not a palindrome and is not a mirrored string
`" -- is a regular palindrome."`	if the string is a palindrome and is not a mirrored string
`" -- is a mirrored string."`	if the string is not a palindrome and is a mirrored string
`" -- is a mirrored palindrome."`	if the string is a palindrome and is a mirrored string

Note that the output line is to include the -'s and spacing exactly as shown in the table above and demonstrated in the Sample Output below.

In addition, after each output line, you must print an empty line.

Sample Input

```
NOTAPALINDROME
ISAPALINILAPASI
2A3MEAS
ATOYOTA
```

Sample Output

```
NOTAPALINDROME -- is not a palindrome.

ISAPALINILAPASI -- is a regular palindrome.

2A3MEAS -- is a mirrored string.

ATOYOTA -- is a mirrored palindrome.
```

Miguel Revilla 2001-04-16

26. 406 - Prime Cuts

Time limit: 3.000 seconds

Prime Cuts

A prime number is a counting number ($1, 2, 3, \ldots$) that is evenly divisible only by 1 and itself. In this problem you are to write a program that will cut some number of prime numbers from the list of prime numbers between (and including) 1 and *N*. Your program will read in a number *N*; determine the list of prime numbers between 1 and*N*; and print the *C**2 prime numbers from the center of the list if there are an even number of prime numbers or (*C**2)-1 prime numbers from the center of the list if there are an odd number of prime numbers in the list.

Input

Each input set will be on a line by itself and will consist of 2 numbers. The first number ($1 \le N \le 1000$) is the maximum number in the complete list of prime numbers between 1 and *N*. The second number ($1 \le C \le N$) defines the *C**2 prime numbers to be printed from the center of the list if the length of the list is even; or the (*C**2)-1 numbers to be printed from the center of the list if the length of the list is odd.

Output

For each input set, you should print the number *N* beginning in column 1 followed by a space, then by the number *C*, then by a colon (`:`), and then by the center numbers from the list of prime numbers as defined above. If the size of the center list exceeds the limits of the list of prime numbers between 1 and *N*, the list of prime numbers between 1 and *N* (inclusive) should be printed. Each number from the center of the list should be preceded by exactly one blank. Each line of output should be followed by a blank line. Hence, your output should follow the exact format shown in the sample output.

Sample Input

```
21 2
18 2
18 18
100 7
```

Sample Output

```
21 2: 5 7 11

18 2: 3 5 7 11

18 18: 1 2 3 5 7 11 13 17

100 7: 13 17 19 23 29 31 37 41 43 47 53 59 61 67
```

27.408 - Uniform Generator

Time limit: 3.000 seconds

Uniform Generator

Computer simulations often require random numbers. One way to generate pseudo-random numbers is via a function of the form

$$seed(x+1) = [seed(x) + STEP] \% MOD$$

where `` $\%$ '' is the modulus operator.

Such a function will generate pseudo-random numbers (*seed*) between 0 and *MOD*-1. One problem with functions of this form is that they will always generate the same pattern over and over. In order to minimize this effect, selecting the *STEP* and *MOD*values carefully can result in a uniform distribution of all values between (and including) 0 and *MOD*-1.

For example, if *STEP* = 3 and *MOD* = 5, the function will generate the series of pseudo-random numbers 0, 3, 1, 4, 2 in a repeating cycle. In this example, all of the numbers between and including 0 and *MOD*-1 will be generated every *MOD* iterations of the function. Note that by the nature of the function to generate the same *seed*(*x*+1) every time *seed*(*x*) occurs means that if a function will generate all the numbers between 0 and *MOD*-1, it will generate pseudo-random numbers uniformly with every*MOD* iterations.

If *STEP* = 15 and *MOD* = 20, the function generates the series 0, 15, 10, 5 (or any other repeating series if the initial seed is other than 0). This is a poor selection of*STEP* and *MOD* because no initial seed will generate all of the numbers from 0 and*MOD*-1.

Your program will determine if choices of *STEP* and *MOD* will generate a uniform distribution of pseudo-random numbers.

Input

Each line of input will contain a pair of integers for *STEP* and *MOD* in that order ($1 \le STEP, MOD \le 10\,000\,000$).

Output

For each line of input, your program should print the *STEP* value right- justified in columns 1 through 10, the *MOD* value right-justified in columns 11 through 20 and either ``Good Choice" or ``Bad Choice" left-justified starting in column 25. The ``Good Choice" message should be printed when the selection of *STEP* and *MOD*will generate all the numbers between and including 0 and *MOD*-1 when MOD numbers are generated. Otherwise, your program should print the message ``Bad Choice". After each output test set, your program should print exactly one blank line.

Sample Input

```
3 5
15 20
63923 99999
```

Sample Output

```
         3         5    Good Choice

        15        20    Bad Choice

     63923     99999    Good Choice
```

28. 409 - Excuses, Excuses!

Time limit: 3.000 seconds

Excuses, Excuses!

Judge Ito is having a problem with people subpoenaed for jury duty giving rather lame excuses in order to avoid serving. In order to reduce the amount of time required listening to goofy excuses, Judge Ito has asked that you write a program that will search for a list of keywords in a list of excuses identifying lame excuses. Keywords can be matched in an excuse regardless of case.

Input

Input to your program will consist of multiple sets of data.

- Line 1 of each set will contain exactly two integers. The first number ($1 \le K \le 20$) defines the number of keywords to be used in the search. The second number ($1 \le E \le 20$) defines the number of excuses in the set to be searched.
- Lines 2 through *K*+1 each contain exactly one keyword.
- Lines *K*+2 through *K*+1+*E* each contain exactly one excuse.
- All keywords in the keyword list will contain only contiguous lower case alphabetic characters of length *L* ($1 \le L \le 20$) and will occupy columns 1 through *L* in the input line.
- All excuses can contain any upper or lower case alphanumeric character, a space, or any of the following punctuation marks [SPMamp".,!?&] not including the square brackets and will not exceed 70 characters in length.
- Excuses will contain at least 1 non-space character.

Output

For each input set, you are to print the worst excuse(s) from the list.

- The worst excuse(s) is/are defined as the excuse(s) which contains the largest number of incidences of keywords.
- If a keyword occurs more than once in an excuse, each occurrance is considered a separate incidence.

- A keyword ``occurs" in an excuse if and only if it exists in the string in contiguous form and is delimited by the beginning or end of the line or any non-alphabetic character or a space.

For each set of input, you are to print a single line with the number of the set immediately after the string ``Excuse Set #". (See the Sample Output). The following line(s) is/are to contain the worst excuse(s) one per line exactly as read in. If there is more than one worst excuse, you may print them in any order.

After each set of output, you should print a blank line.

Sample Input

```
5 3
dog
ate
homework
canary
died
My dog ate my homework.
Can you believe my dog died after eating my canary... AND MY HOMEWORK?
This excuse is so good that it contain 0 keywords.
6 5
superhighway
crazy
thermonuclear
bedroom
war
building
I am having a superhighway built in my bedroom.
I am actually crazy.
1234567890.....,,,,,0987654321?????!!!!!!
There was a thermonuclear war!
I ate my dog, my canary, and my homework ... note outdated keywords?
```

Sample Output

```
Excuse Set #1
Can you believe my dog died after eating my canary... AND MY HOMEWORK?

Excuse Set #2
I am having a superhighway built in my bedroom.
There was a thermonuclear war!
```

29. 412 - Pi

Time limit: 3.000 seconds

Professor Robert A. J. Matthews of the Applied Mathematics and Computer Science Department at the University of Aston in Birmingham, England has recently described howthe positions of stars across the night skymay be used to deduce a surprisingly accurate value of π. This result followed from the application of certain theorems in number theory.

Here, we don'thave the night sky, but can use the same theoretical basis to form an estimate for π:

Given any pair of whole numbers chosen from a large, random collection of numbers, the probability that the twonumbers have no common factor other than one (1) is

$$\frac{6}{\pi^2}$$

For example, using the *small* collection of numbers: 2, 3, 4, 5, 6; there are 10 pairs that can be formed: (2,3), (2,4), etc. Six of the 10 pairs: (2,3), (2,5), (3,4), (3,5), (4,5) and (5,6) have no common factor other than one. Using the ratio of the counts as the probability we have:

$$\frac{6}{\pi^2} \approx \frac{6}{10}$$

$$\pi \approx 3.162$$

In this problem, you'll receive a series of data sets. Each data set contains a set of pseudo-random positive integers. For each data set, find the portion of the pairs which may be formed that have nocommon factor other than one (1), and use the method illustrated above to obtain an estimate for π. Report this estimate for each data set.

Input

The input consists of a series of data sets.

The first line of each data set contains a positive integer value, *N*, greater than one (1) and less than 50.

There is one positive integer per line for the next N lines that constitute the set for which the pairs are to be examined. These integers are each greater than 0 and less than 32768.

Each integer of the input stream has its first digit as the first character on the input line.

The set size designator, N, will be zero to indicate the end of data.

Output

A line with a single real value is to be emitted for each input data set encountered. This value is the estimate for π for the data set. An output format like the sample below should be used. Answers must be rounded to six digits after the decimal point.

For some data sets, it may be impossible to estimate a value for π. This occurs when there are *no* pairs without common factors. In these cases, emit the single-line message:

```
No estimate for this data set.
```

exactly, starting with the first character, `"N"`, as the first character on the line.

Sample Input

```
5
2
3
4
5
6
2
13
39
0
```

Sample Output (Your output for the float/real, using your chosen language, may be default-formatted differently).

```
3.162278
No estimate for this data set.
```

30. 424 - Integer Inquiry

Time limit: 3.000 seconds

Integer Inquiry

One of the first users of BIT's new supercomputer was Chip Diller. He extended his exploration of powers of 3 to go from 0 to 333 and he explored taking various sums of those numbers.

``This supercomputer is great,'' remarked Chip. ``I only wish Timothy were here to see these results.'' (Chip moved to a new apartment, once one became available on the third floor of the Lemon Sky apartments on Third Street.)

Input

The input will consist of at most 100 lines of text, each of which contains a single VeryLongInteger. Each VeryLongInteger will be 100 or fewer characters in length, and will only contain digits (no VeryLongInteger will be negative).

The final input line will contain a single zero on a line by itself.

Output

Your program should output the sum of the VeryLongIntegers given in the input.

Sample Input

```
123456789012345678901234567890
123456789012345678901234567890
123456789012345678901234567890
0
```

Sample Output

```
370370367037037036703703703670
```

31. 435 - Block Voting

Time limit: 3.000 seconds

Block Voting

Different types of electoral systems exist. In a block voting system the members of a party do not vote individually as they like, but instead they must collectively accept or reject a proposal. Although a party with many votes clearly has more power than a party with few votes, the votes of a small party can nevertheless be crucial when they are needed to obtain a majority. Consider for example the following five-party system:

party	votes
A	7
B	4
C	2
D	6
E	6

Coalition {A,B} has 7 + 4 = 11 votes, which is not a majority. When party C joins coalition {A,B}, however, {A,B,C} becomes a winning coalition with 7+4+2 = 13 votes. So even though C is a small party, it can play an important role.

As a measure of a party's power in a block voting system, John F. Banzhaf III proposed to use the *power index*. The key idea is that a party's power is determined by the number of minority coalitions that it can join and turn into a (winning) majority coalition. Note that the empty coalition is also a minority coalition and that a coalition only forms a majority when it has more than half of the total number of votes. In the example just given, a majority coalition must have at least 13 votes.

In an ideal system, a party's power index is proportional to the number of members of that party.

Your task is to write a program that, given an input as shown above, computes for each party its power index.

Input Specification

The first line contains a single integer which equals the number of test cases that follow. Each of the following lines contains one test case.

The first number on a line contains an integer *P* in [1...20] which equals the number of parties for that test case. This integer is followed by *P* positive integers, separated by spaces. Each of these integers represents the number of members of a party in the electoral system. The *i*-th number represents party number *i*. No electoral system has more than 1000 votes.

Output Specification

For each test case, you must generate *P* lines of output, followed by one empty line. *P*is the number of parties for the test case in question. The *i*-th line (*i* in [1...*P*]) contains the sentence:

```
party i has power index I
```

where *I* is the power index of party *i*.

Sample Input

```
3
5 7 4 2 6 6
6 12 9 7 3 1 1
3 2 1 1
```

Sample Output

```
party 1 has power index 10
party 2 has power index 2
party 3 has power index 2
party 4 has power index 6
party 5 has power index 6

party 1 has power index 18
party 2 has power index 14
party 3 has power index 14
party 4 has power index 2
party 5 has power index 2
party 6 has power index 2

party 1 has power index 3
party 2 has power index 1
party 3 has power index 1
```

32. 440 - Eeny Meeny Moo

Time limit: 3.000 seconds

Eeny Meeny Moo

Surely you have made the experience that when too many people use the Internet simultaneously, the net becomes very, very slow.

To put an end to this problem, the University of Ulm has developed a contingency scheme for times of peak load to cut off net access for some cities of the country in a systematic, totally fair manner. Germany's cities were enumerated randomly from 1 to*n*. Freiburg was number 1, Ulm was number 2, Karlsruhe was number 3, and so on in a purely random order.

Then a number *m* would be picked at random, and Internet access would first be cut off in city 1 (clearly the fairest starting point) and then in every *m*th city after that, wrapping around to 1 after *n*, and ignoring cities already cut off. For example, if *n*=17 and *m*=5, net access would be cut off to the cities in the order [1,6,11,16,5,12,2,9,17,10,4,15,14,3,8,13,7]. The problem is that it is clearly fairest to cut off Ulm last (after all, this is where the best programmers come from), so for a given n, the random number m needs to be carefully chosen so that city 2 is the last city selected.

Your job is to write a program that will read in a number of cities *n* and then determine the smallest integer *m* that will ensure that Ulm can surf the net while the rest of the country is cut off.

Input Specification

The input file will contain one or more lines, each line containing one integer *n* with $3 \le n < 150$, representing the number of cities in the country.

Input is terminated by a value of zero (0) for *n*.

Output Specification

For each line of the input, print one line containing the integer *m* fulfilling the requirement specified above.

Sample Input

```
3
4
5
6
7
8
9
10
11
12
0
```

Sample Output

```
2
5
2
4
3
11
2
3
8
16
```

33. 445 - Marvelous Mazes

Time limit: 3.000 seconds

Marvelous Mazes

Your mission, if you decide to accept it, is to create a maze drawing program. A maze will consist of the alphabetic characters A-Z, * (asterisk), and spaces.

Input and Output

Your program will get the information for the mazes from the input file. This file will contain lines of characters which your program must interpret to draw a maze. Each row of the maze will be described by a series of numbers and characters, where the numbers before a character tell how many times that character will be used. If there are multiple digits in a number before a character, then the number of times to repeat the character is the sum of the digits before that character.

The lowercase letter "b" will be used in the input file to represent spaces in the maze. The descriptions for different rows in the maze will be separated by an exclamation point (!) or by an end of line.

Descriptions for different mazes will be separated by a blank line in both input and output. The input file will be terminated by an end of file.

There is no limit to the number of rows in a maze or the number of mazes in a file, though no row will contain more than 132 characters.

Happy mazing!

Sample Input

```
1T1b5T!1T2b1T1b2T!1T1b1T2b2T!1T3b1T1b1T!3T3b1T!1T3b1T1b1T!5T1*1T

11X21b1X
4X1b1X
```

Sample Output

```
T TTTTT
T  T TT
T T  TT
T   T T
TTT   T
T   T T
TTTTT*T

XX   X
XXXX X
```

34. 446 - Kibbles "n" Bits "n" Bits "n" Bits

Time limit: 3.000 seconds

Kibbles `n' Bits `n' Bits `n' Bits

A certain frazzled programmer is writing a program that receives two numbers at a time in hexadecimal form, performs an addition or subtraction on them, and outputs the result in decimal form. However, the binary representation of the hexadecimal numbers also needs to be output, in the exact format shown by the sample output below.

This programmer would gladly write the routine to do this himself, but every time he tries to do anything in base 2, he breaks out in hives. So if you write this little routine for him, he would be eternally grateful.

You may assume the following:

- The largest allowable hexadecimal number is FFF.
- When subtracting, the second number will always be smaller than the first, i.e. no negative results.
- The spacing in the input file will be uniform throughout, i.e. no spaces at the beginning of a line, and one space between each element.

Input

The input for this program will come from a file. The format for the file is as follows:
N (This is the number of expressions to compute)
*HEX*1 (+ or -) *HEX*2 (The first expression)
.
.
.
*HEX*1 (+ or -) *HEX*2 (The *n*th expression)

Output

The output file should be in the following format:

*BINARY*1 (+ or -) *BINARY*2 = *DECIMAL* (first result)

.
.
.

*BINARY*1 (+ or -) *BINARY*2 = *DECIMAL* (*n*th result)

Sample Input

```
2
A + 3
AAA + BBB
```

Sample Output

```
0000000001010 + 0000000000011 = 13
0101010101010 + 0101110111011 = 5733
```

35. 463 - Polynomial Factorization

Time limit: 3.000 seconds

Polynomial Factorization

A polynomial is of degree k if the largest power of the variable in any term is no greter than k. For example,

$$x^2 + 4x + 2$$

is of degree 2. It is also of degree 3, 4, 5, ...

A polynomial with integer coefficients is "prime" if it cannot be expressed as the product of two lower-degree polynomials with integer coefficients.

Write a program to express a 4th degree polynomial with integer coefficients as the product of one or more prime polynomials.

The sample below tell us that the polynomial $2x^4 + 7x^3 - x^2 - 6x - 2$ has prime factors x-1, 2x+1 and $x^2 + 4x + 2$.

Input

The input consists on several test cases, one on each line. Each test case consists on 5 integers, representing the coefficients of a degree 4 polynomial in decreasing order of the variable power from 4 to 0. The first number will never be negative or 0.

Output

For each test case, print the integer coefficients of each prime factor on a single line, in decreasing order of variable power. The first coefficient of prime polynomials should be positive, except maybe in the last polynomial on the list. The polynomials must be printed in increasing order of degree, and when two or more degrees match they must be sorted by increasing value of coefficients from the greatest to the lowest degree. Extra factors should be added to the last polynomial on the list. Print a blank line after each test case.

Sample Input

```
2 7 -1 -6 -2
2 0 0 0 0
```

Sample Output

```
1 -1
2 1
1 4 2

1 0
1 0
1 0
2 0
```

36. 468 - Key to Success

Time limit: 3.000 seconds

Key to Success

Any one-to-one mapping, *f*, of any alphabet to itself can be used to encode text by replacing each occurrence of any letter, *c*, with *f*(*c*). One such mapping could be the mapping of a letter to three positions beyond the letter in the alphabet. That is, $a \to d$,$b \to e$, $c \to f$, $d \to g$ and so on.

With this mapping, ``The car is blue" will be encoded as ``Wkh fdu lv eoxh".

Input and Output

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Your correct program should decodes the contents of each input set according to the following guidelines:

1. Only letters are encoded. Letters are mapped to letters. Uppercase letters are different from their lowercase counter parts.
2. The mapping that defines the encoding is one-to-one. That is, two different letters never map to the same letter of the alphabet ($a \to x$ and $t \to x$ is impossible).
3. There are two input lines - the first one contains a text (not encoded) and the second one contains an encoded text. This text is to be decoded by your program.
4. Both lines are written by the same person.
5. It is to be assumed that any person uses letters of the alphabet with the same**RELATIVE FREQUENCY** from document to document and no two letters are used with the same frequency. That is, the most frequently used letter in the first line maps to the most frequently used letter in the second one; the second most frequently used letter maps to the second most frequently used letter and so on.

The outputs of two consecutive cases will be separated by a blank line.

Sample Intput

```
1

abacxbacac
qqqqqrrrrssstt
```

Sample Output

```
aaaaaccccbbbxx
```

37. 484 - The Department of Redundancy Department

Time limit: 3.000 seconds

The Department of Redundancy Department

Write a program that will remove all duplicates from a sequence of integers and print the list of unique integers occuring in the input sequence, along with the number of occurences of each.

Input

The input file will contain a sequence of integers (positive, negative, and/or zero). The input file may be arbitrarily long.

Output

The output for this program will be a sequence of ordered pairs, separated by newlines. The first element of the pair must be an integer from the input file. The second element must be the number of times that that particular integer appeared in the input file. The elements in each pair are to be separated by space characters. The integers are to appear in the order in which they were contained in the input file.

Sample Input

```
3 1 2 2 1 3 5 3 3 2
```

Sample Output

```
3 4
1 2
2 3
5 1
```

38. 488 - Triangle Wave

Time limit: 3.000 seconds

Triangle Wave

In this problem you are to generate a triangular wave form according to a specified pair of Amplitude and Frequency.

Input and Output

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Each input set will contain two integers, each on a separate line. The first integer is the Amplitude; the second integer is the Frequency.

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

For the output of your program, you will be printing wave forms each separated by a blank line. The total number of wave forms equals the Frequency, and the horizontal ``height" of each wave equals the Amplitude. The Amplitude will never be greater than nine.

The waveform itself should be filled with integers on each line which indicate the ``height" of that line.

NOTE: There is a blank line after each separate waveform, **excluding** the last one.

Sample Input

```
1

3
2
```

Sample Output

```
1
22
333
22
1

1
22
333
22
1
```

39. 489 - Hangman Judge

Time limit: 3.000 seconds

In ``Hangman Judge,'' you are to write a program that judges a series of Hangman games. For each game, the answer to the puzzle is given as well as the guesses. Rules are the same as the classic game of hangman, and are given as follows:

1. The contestant tries to solve to puzzle by guessing one letter at a time.
2. Every time a guess is correct, all the characters in the word that match the guess will be ``turned over.'' For example, if your guess is ``o'' and the word is ``book'', then both ``o''s in the solution will be counted as ``solved.''
3. Every time a wrong guess is made, a stroke will be added to the drawing of a hangman, which needs 7 strokes to complete. Each unique wrong guess only counts against the contestant once.

```
4.      ______
5.      |   |
6.      |   O
7.      | /|\
8.      |   |
9.      | / \
10.   __|_
11.   |   |______
      |__________|
```

12. If the drawing of the hangman is completed before the contestant has successfully guessed all the characters of the word, the contestant loses.
13. If the contestant has guessed all the characters of the word before the drawing is complete, the contestant wins the game.
14. If the contestant does not guess enough letters to either win or lose, the contestant chickens out.

Your task as the ``Hangman Judge'' is to determine, for each game, whether the contestant wins, loses, or fails to finish a game.

Input

Your program will be given a series of inputs regarding the status of a game. All input will be in lower case. The first line of each section will contain a number to indicate which round of the game is being played; the next line will be the solution to the puzzle; the last line is a sequence of the guesses made by the contestant. A round number of `-1` would indicate the end of all games (and input).

Output

The output of your program is to indicate which round of the game the contestant is currently playing as well as the result of the game. There are three possible results:

```
You win.
You lose.
You chickened out.
```

Sample Input

```
1
cheese
chese
2
cheese
abcdefg
3
cheese
abcdefgij
-1
```

Sample Output

```
Round 1
You win.
Round 2
You chickened out.
Round 3
You lose.
```

40. 490 - Rotating Sentences

Time limit: 3.000 seconds

Rotating Sentences

In ``Rotating Sentences," you are asked to rotate a series of input sentences 90 degrees clockwise. So instead of displaying the input sentences from left to right and top to bottom, your program will display them from top to bottom and right to left.

Input and Output

As input to your program, you will be given a maximum of 100 sentences, each not exceeding 100 characters long. Legal characters include: newline, space, any punctuation characters, digits, and lower case or upper case English letters. (NOTE: Tabs are not legal characters.)

The output of the program should have the last sentence printed out vertically in the leftmost column; the first sentence of the input would subsequently end up at the rightmost column.

Sample Input

```
Rene Decartes once said,
"I think, therefore I am."
```

Sample Output

```
"R
Ie
 n
te
h
iD
ne
kc
,a
 r
tt
he
es
r
eo
fn
oc
re
e
 s
Ia
 i
ad
m,
.
"
```

41. 495 - Fibonacci Freeze

Time limit: 3.000 seconds

Fibonacci Freeze

The Fibonacci numbers (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...) are defined by the recurrence:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2} \text{ for all } i \geq 2 \end{aligned}$$

Write a program to calculate the Fibonacci Numbers.

Input and Output

The input to your program would be a sequence of numbers smaller or equal than 5000, each on a separate line, specifying which Fibonacci number to calculate.

Your program should output the Fibonacci number for each input value, one per line.

Sample Input

```
5
7
11
```

Sample Output

```
The Fibonacci number for 5 is 5
The Fibonacci number for 7 is 13
The Fibonacci number for 11 is 89
```

42. 498 - Polly the Polynomial

Time limit: 3.000 seconds

Polly the Polynomial

Algebra! Remember algebra? There is a theory that as engineers progresses further and further in their studies, they lose basic math skills. This problem is designed to help you remember those *basic* algebra skills, make the world a better place, etc., etc.

Input

Your program should accept an even number of lines of text. Each pair of lines will represent one problem. The first line will contain a list of integers { $c_0, c_1, ..., c_n$ } which represent a set of coefficients to a polynomial expression. The order of the polynomial is *n*. The coefficients should be paired with the terms of the polynomial in the following manner:

$$c_0x^n + c_1x^{n-1} + ... + c_nx^0$$

The second line of text represents a sequence of values for *x*, { $x_0, x_1, ..., x_m$ }.

Output

For each pair of lines, your program should evaluate the polynomial for all the values of*x* (x_0 through x_m) and output the resulting values on a single line.

Sample Input

```
-2
5 0 1 6
1 -1
7 6 -1
```

Sample Output

```
-2 -2 -2 -2
6 5 -2
```

43. 499 - What's The Frequency, Kenneth?

Time limit: 3.000 seconds

What's The Frequency, Kenneth?

```
#include <stdio.h>

main()
{
  int i;
  char *suffix[]= { "st", "nd", "rd" };
  char *item[]= { "Unix" , "cat", "sed", "awk", "grep", "ed", "vi"};

  printf("In the beginning, there was nothing.\n");
  for (i= 0; i < 7; i++)
    printf("And on the %d%s day, God created %s. And it was good.\n",
           i + 1, (i < 3) ? suffix[i] : "th", item[i]);
}
```

But then God saw that `vi` led people into temptation. Instead of choosing the righteous ways of `make`, `dbx`, and `RCS`, people used long command lines, `printf()`, and tape backups.

So God decreed, ``I see that Engineers have thus defiled my `vi`. And so, I shall create*emacs*, an editor more powerful than words. Further, for each instantiation `vi` hitherto, the Engineer responsible shalt perform Penance. And lo, the Penance wilt be painful; there will be much wailing and gnushingof teeth. The Engineer will read many lines of text. For each line of text, the Engineer must tell me which letters occur the most frequently."

``I charge you all with My Golden Rule: 'Friends shalt not let friends use `vi`'."

Input and Output

Each line of output should contain a list of letters that all occured with the highest frequency in the corresponding input line, followed by the frequency.

The list of letters should be an alphabetical list of upper case letters followed by an alphabetical list of lower case letters.

Sample Input

```
When riding your bicycle backwards down a one-way street, if the
wheel falls of a canoe, how many ball bearings does it take to fill
up a water buffalo?
Hello Howard.
```

Sample Output

```
e 6
al 7
a 3
Hlo 2
```

44. 530 - Binomial Showdown

Time limit: 3.000 seconds

Binomial Showdown

In how many ways can you choose *k* elements out of *n* elements, not taking order into account?

Write a program to compute this number.

Input Specification

The input file will contain one or more test cases.

Each test case consists of one line containing two integers *n* ($n \geq 1$) and *k* ($0 \leq k \leq n$).

Input is terminated by two zeroes for *n* and *k*.

Output Specification

For each test case, print one line containing the required number. This number will always fit into an integer, i.e. it will be less than 2^{31}.

Warning: Don't underestimate the problem. The result will fit into an integer - but if all intermediate results arising during the computation will also fit into an integer depends on your algorithm. The test cases will go to the limit.

Sample Input

```
4 2
10 5
49 6
0 0
```

Sample Output

```
6
252
13983816
```

45. 541 - Error Correction

Time limit: 3.000 seconds

Error Correction

A boolean matrix has the *parity property* when each row and each column has an even sum, i.e. contains an even number of bits which are set. Here's a 4 x 4 matrix which has the parity property:

```
1 0 1 0
0 0 0 0
1 1 1 1
0 1 0 1
```

The sums of the rows are 2, 0, 4 and 2. The sums of the columns are 2, 2, 2 and 2.

Your job is to write a program that reads in a matrix and checks if it has the parity property. If not, your program should check if the parity property can be established by changing only one bit. If this is not possible either, the matrix should be classified as corrupt.

Input

The input file will contain one or more test cases. The first line of each test case contains one integer *n* (*n*<100), representing the size of the matrix. On the next *n* lines, there will be *n* integers per line. No other integers than 0 and 1 will occur in the matrix. Input will be terminated by a value of 0 for *n*.

Output

For each matrix in the input file, print one line. If the matrix already has the parity property, print ``OK". If the parity property can be established by changing one bit, print ``Change bit (*i*,*j*)" where *i* is the row and *j* the column of the bit to be changed. Otherwise, print ``Corrupt".

Sample Input

```
4
1 0 1 0
0 0 0 0
1 1 1 1
0 1 0 1
4
1 0 1 0
0 0 1 0
1 1 1 1
0 1 0 1
4
1 0 1 0
0 1 1 0
1 1 1 1
0 1 0 1
0
```

Sample Output

```
OK
Change bit (2,3)
Corrupt
```

46. 543 - Goldbach's Conjecture

Time limit: 3.000 seconds

Goldbach's Conjecture

In 1742, Christian Goldbach, a German amateur mathematician, sent a letter to Leonhard Euler in which he made the following conjecture:

Every number greater than 2 can be written as the sum of three prime numbers.

Goldbach cwas considering 1 as a primer number, a convention that is no longer followed. Later on, Euler re-expressed the conjecture as:

Every even number greater than or equal to 4 can be expressed as the sum of two prime numbers.

For example:

- 8 = 3 + 5. Both 3 and 5 are odd prime numbers.
- 20 = 3 + 17 = 7 + 13.
- 42 = 5 + 37 = 11 + 31 = 13 + 29 = 19 + 23.

Today it is still unproven whether the conjecture is right. (Oh wait, I have the proof of course, but it is too long to write it on the margin of this page.)

Anyway, your task is now to verify Goldbach's conjecture as expressed by Euler for all even numbers less than a million.

Input

The input file will contain one or more test cases.

Each test case consists of one even integer *n* with $6 \le n < 1000000$.

Input will be terminated by a value of 0 for *n*.

Output

For each test case, print one line of the form *n* = *a* + *b*, where *a* and *b* are odd primes. Numbers and operators should be separated by exactly one blank like in the sample output below. If there is more than one pair of odd primes adding up to *n*, choose the pair where the difference *b* - *a* is maximized.

If there is no such pair, print a line saying ``Goldbach's conjecture is wrong.''

Sample Input

```
8
20
42
0
```

Sample Output

```
8 = 3 + 5
20 = 3 + 17
42 = 5 + 37
```

Miguel A. Revilla
1999-01-11

47. 575 - Skew Binary

Time limit: 3.000 seconds

Skew Binary

When a number is expressed in decimal, the *k*-th digit represents a multiple of 10^k. (Digits are numbered from right to left, where the least significant digit is number 0.) For example,

$$81307_{10} = 8\times10^4+1\times10^3+3\times10^2+0\times10^1+7\times10^0 = 80000+1000+300+0+7 = 81307.$$

When a number is expressed in binary, the *k*-th digit represents a multiple of 2^k. For example,

$$10011_2 = 1\times2^4+0\times2^3+0\times2^2+1\times2^1+1\times2^0 = 16+0+0+2+1 = 19.$$

In **skew binary**, the *k*-th digit represents a multiple of $2^{k+1} - 1$. The only possible digits are 0 and 1, except that the least-significant nonzero digit can be a 2. For example,

$$10120_{skew} = 1\times(2^5-1)+0\times(2^4-1)+1\times(2^3-1)+2\times(2^2-1)+0\times(2^1-1) = 31+0+7+6+0 = 44.$$

The first 10 numbers in skew binary are 0, 1, 2, 10, 11, 12, 20, 100, 101, and 102. (Skew binary is useful in some applications because it is possible to add 1 with at most one carry. However, this has nothing to do with the current problem.)

Input

The input file contains one or more lines, each of which contains an integer *n*. If *n* = 0 it signals the end of the input, and otherwise *n* is a nonnegative integer in skew binary.

Output

For each number, output the decimal equivalent. The decimal value of *n* will be at most2^{31} - 1 = 2147483647.

Sample Input

```
10120
200000000000000000000000000000
10
1000000000000000000000000000000
11
100
11111000001110000101101102000
0
```

Sample Output

```
44
2147483646
3
2147483647
4
7
1041110737
```

Miguel A. Revilla
1998-03-10

48. 579 - ClockHands

Time limit: 3.000 seconds

ClockHands

The medieval interest in mechanical contrivances is well illustrated by the development of the mechanical clock, the oldest of which is driven by weights and controlled by a verge, an oscillating arm engaging with a gear wheel. It dates back to 1386.

Clocks driven by springs had appeared by the mid-15th century, making it possible to con- struct more compact mechanisms and preparing the way for the portable clock.

English spring-driven pendulum clocks were first commonly kept on a small wall bracket and later on a shelf. Many bracket clocks contained a drawer to hold the winding key. The earliest bracket clocks, made for a period after 1660, were of architectural design, with pillars at the sides and a pediment on top.

In 17th- and 18th-century France, the table clock became an object of monumental design, the best examples of which are minor works of sculpture.

The longcase clocks (also called grandfather clocks) are tall pendulum clock enclosed in a wooden case that stands upon the floor and is typically from 6 to 7.5 feet (1.8 to 2.3 m) in height. Later, the name ``grandfather clock" became popular after the popular song "My Grandfather's Clock," written in 1876 by Henry Clay Work.

One of the first atomic clocks was an ammonia-controlled clock. It was built in 1949 at the National Bureau of Standards, Washington, D.C.; in this clock the frequency did not vary by more than one part in 10^8

Nuclear clocks are built using two clocks. The aggregate of atoms that emit the gamma radiation of precise frequency may be called the emitter clock; the group of atoms that absorb this radiation is the absorber clock. One pair of these nuclear clocks can detect energy changes of one part in 10^{14} , being about 1,000 times more sensitive than the best atomic clock.

The cesium clock is the most accurate type of clock yet developed. This device makes use of transitions between the spin states of the cesium nucleus and produces a frequency which is so regular that it has been adopted for establishing the time standard.

The history of clocks is fascinating, but unrelated to this problem. In this problem, you are asked to find the angle between the minute hand and the hour hand on a regular analog clock. Assume that the second hand, if there were one, would be pointing straight up at the 12. Give all angles as the smallest positive angles. For example 9:00 is 90 degrees; not -90 or 270 degrees.

Input

The input is a list of times in the form *H*:*M*, each on their own line, with $1 \le H \le 12$ and $00 \le M \le 59$. The input is terminated with the time `0:00`. Note that *H* may be represented with 1 or 2 digits (for 1-9 or 10-12, respectively); *M*is always represented with 2 digits (The input times are what you typically see on a digital clock).

Output

The output displays the smallest positive angle in degrees between the hands for each time. The answer should between `0` degrees and `180` degrees for all input times. Display each angle on a line by itself in the same order as the input. The output should be rounded to the nearest 1/1000, i.e., three places after the decimal point should be printed.

Sample Input

```
12:00
9:00
8:10
0:00
```

Sample Output

```
0.000
90.000
175.000
```

Miguel A. Revilla
1998-03-10

49. 583 - Prime Factors

Time limit: 3.000 seconds

Prime Factors

Webster defines *prime* as:

prime (prim) *n*.[**ME**, fr. **MF**, fem. of *prin* first, **L** *primus*; akin to **L** *prior*] **1** :first in time: **original** **2** **a** : having no factor except itself and one ⟨ 3 is a number ⟩ **b** : having no common factor except one ⟨ 12 and 25 are relatively ⟩ **3** **a** : first in rank, authority or significance : **principal** **b** : having the highest quality or value ⟨ television time ⟩ [from *Webster's New Collegiate Dictionary*]

The most relevant definition for this problem is 2a: An integer $g>1$ is said to be *prime* if and only if its only positive divisors are itself and one (otherwise it is said to be*composite*). For example, the number 21 is composite; the number 23 is prime. Note that the decompositon of a positive number g into its prime factors, i.e.,

$$g = f_1 \times f_2 \times \ldots \times f_n$$

is unique if we assert that $f_i > 1$ for all i and $f_i \leq f_j$ for $i<j$.

One interesting class of prime numbers are the so-called *Mersenne* primes which are of the form $2^p - 1$. Euler proved that $2^{31} - 1$ is prime in 1772 -- all without the aid of a computer.

Input

The input will consist of a sequence of numbers. Each line of input will contain one number g in the range $-2^{31} < g < 2^{31}$, but different of -1 and 1. The end of input will be indicated by an input line having a value of zero.

Output

For each line of input, your program should print a line of output consisting of the input number and its prime factors. For an input number $g > 0, g = f_1 \times f_2 \times \ldots \times f_n$, where each f_i is a prime number greater than unity (with $f_i \leq f_j$ for $i<j$), the format of the output line should be

g = f_1 x f_2 x ... x f_n

When $g < 0$, if $|g| = f_1 \times f_2 \times \ldots \times f_n$, the format of the output line should be

g = -1 x f_1 x f_2 x ... x f_n

Sample Input

```
-190
-191
-192
-193
-194
195
196
197
198
199
200
0
```

Sample Output

```
-190 = -1 x 2 x 5 x 19
-191 = -1 x 191
-192 = -1 x 2 x 2 x 2 x 2 x 2 x 2 x 3
-193 = -1 x 193
-194 = -1 x 2 x 97
195 = 3 x 5 x 13
196 = 2 x 2 x 7 x 7
197 = 197
198 = 2 x 3 x 3 x 11
199 = 199
200 = 2 x 2 x 2 x 5 x 5
```

Miguel Revilla
2000-05-19

50. 673 - Parentheses Balance

Time limit: 3.000 seconds

Parentheses Balance

You are given a string consisting of parentheses `()` and `[]`. A string of this type is said to be *correct*:

(a)

if it is the empty string

(b)

if A and B are correct, AB is correct,

(c)

if A is correct, `(A)` and `[A]` is correct.

Write a program that takes a sequence of strings of this type and check their correctness. Your program can assume that the maximum string length is 128.

Input

The file contains a positive integer *n* and a sequence of *n* strings of parentheses `()` and`[]`, one string a line.

Output

A sequence of `Yes` or `No` on the output file.

Sample Input

```
3
([])
(([()])))
([()[]()])()
```

Sample Output

```
Yes
No
Yes
```

Miguel Revilla
2000-08-14

51. 686 - Goldbach's Conjecture (II)

Time limit: 3.000 seconds

Goldbach's Conjecture (II)

Goldbach's Conjecture: For any even number n greater than or equal to 4, there exists at least one pair of prime numbers p_1 and p_2 such that $n = p_1 + p_2$.

This conjecture has not been proved nor refused yet. No one is sure whether this conjecture actually holds. However, one can find such a pair of prime numbers, if any, for a given even number. The problem here is to write a program that reports the number of all the pairs of prime numbers satisfying the condition in the conjecture for a given even number.

A sequence of even numbers is given as input. Corresponding to each number, the program should output the number of pairs mentioned above. Notice that we are interested in the number of essentially different pairs and therefore you should not count(p_1, p_2) and (p_2, p_1) separately as two different pairs.

Input

An integer is given in each input line. You may assume that each integer is even, and is greater than or equal to 4 and less than 2^{15}. The end of the input is indicated by a number 0.

Output

Each output line should contain an integer number. No other characters should appear in the output.

Sample Input

```
6
10
12
0
```

Sample Output

```
1
2
1
```

Miguel A. Revilla
1999-03-05

52. 693 - Digital Racing Circuit

Time limit: 3.000 seconds

Digital Racing Circuit

You have an ideally small racing car on an *x*-*y* plane ($0 \leq x, y \leq 255$, where bigger y denotes upper coordinate). The racing circuit course is figured by two solid walls. Each wall is a closed loop of connected line segments. End point coordinates of every line segment are both integers (See Figure 1). Thus, a wall is represented by a list of end point integer coordinates (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). The start line and the goal line are identical.

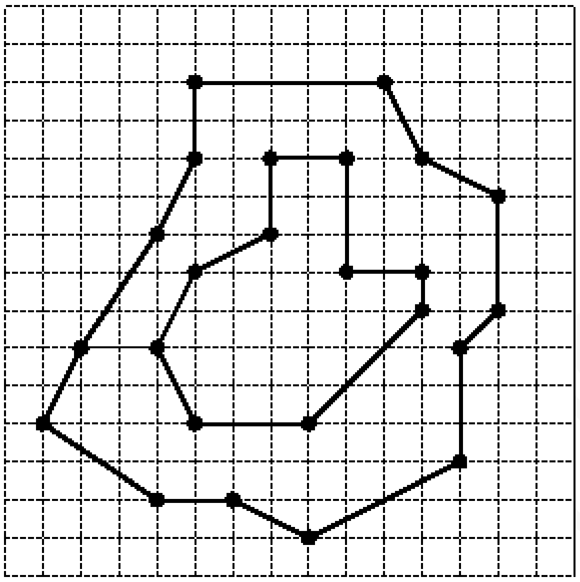


Figure 1. A Simple Course

For the qualification run, you start the car at any integer coordinate position on the start line, say (s_x , s_y).

At any clock $t(\geq 0)$, according to the acceleration parameter at *t*, $(a_{x,t}, a_{y,t})$, the velocity changes instantly to $(v_{x,t-1} + a_{x,t}, v_{y,t-1} + a_{y,t})$, if the velocity at *t* - 1 is $(v_{x,t-1}, v_{y,t-1})$. The velocity will be kept constant until the next clock. It is assumed that the velocity at clock -1, $(v_{x,-1}, v_{y,-1})$ is (0, 0). Each of the acceleration components must be either -1, 0, or 1, because your car does not have so fantastic engine and brake. In other words, any successive pair of velocities should not differ more than 1 for either*x*-component or *y*-component. Note that your trajectory will be piecewise linear as the walls are.

Your car should not touch nor run over the circuit wall, or your car will be crashed, even at the start line. The referee watches your car's trajectory very carefully, checking whether or not the trajectory touches the wall or attempts to cross the wall.

The objective of the qualification run is to finish your run within as few clock units as possible, without suffering from any interference by other racing cars. That is, you run alone the circuit around clockwise and reach, namely touch or go across the goal line without having been crashed. Don't be crashed even in the last trajectory segment after you reach the goal line. But we don't care whatever happens after that clock.

Your final lap time is the clock *t* when you reach the goal line for the first time after you have once left the start line. But it needs a little adjustment at the goal instant. When you cross the goal line, only the necessary fraction of your car's last trajectory segment is counted. For example, if the length of your final trajectory segment is 3 and only its 1/3 fraction is needed to reach the goal line, you have to add only 0.333 instead of 1 clock unit.

Drivers are requested to control their cars as cleverly as possible to run fast but avoiding crash. ALAS! The racing committee decided that it is too DANGEROUS to allow novices to run the circuit. In the last year, it was reported that some novices wrenched their fingers by typing too enthusiastically their programs. So, this year, you are invited as a referee assistant in order to accumulate the experience on this dangerous car race.

A number of experienced drivers are now running the circuit for the qualification for semi-finals. They submit their driving records to the referee. The referee has to check the records one by one whether it is not a fake.

Now, you are requested to make a program to check the validity of driving records for any given course configuration. Is the start point right on the start line without touching the walls? Is every value in the acceleration parameter list either one of -1, 0, and 1? Does the length of acceleration parameter list match the reported lap time? That is, aren't there any excess acceleration parameters after the goal, or are these enough to reach the goal? Doesn't it involve any crash? Does it mean a clockwise running all around? (Note that it is not inhibited to run backward temporarily unless crossing the start line again.) Is the lap time calculation correct? You should allow a rounding error up to 0.01 clock unit in the lap time.

Input

The input consists of several courses and the first line of input is number of courses.

A course configuration is given by two lists representing the inner wall and the outer wall, respectively. Each list shows the end point coordinates of the line segments that comprise the wall. A course configuration looks as follows:

$i_{x,1}\ i_{y,1} \dots i_{x,N}\ i_{y,N}$ 99999

$o_{x,1}\ o_{y,1} \dots o_{x,M}\ o_{y,M}$ 99999

where data are alternating *x*-coordinates and *y*-coordinates that are all non-negative integers (≤ 255) terminated by 99999. The start/goal line is a line segment connecting the coordinates ($i_{x,1}$, $i_{y,1}$) and ($o_{x,1}$, $o_{y,1}$). For simplicity, $i_{y,1}$ is assumed to be equal to $o_{y,1}$; that is, the start/goal line is horizontal on the *x*-*y* plane. Note that *N*and *M* may vary among course configurations, but do not exceed 100, because too many curves involve much danger even for experienced drivers. You need not check the validity of the course configuration.

A driving record consists of three parts, which is terminated by 99999: two integers s_x,s_y for the start position (s_x, s_y), the lap time with three digits after the decimal point, and the sequential list of acceleration parameters at all clocks until the goal. It is assumed that the length of the acceleration parameter list does not exceed 500. A driving record looks like the following:

$s_x\ s_y$

lap-time

$a_{x,0}\ a_{y,0}\ a_{x,1}\ a_{y,1} \dots a_{x,L}\ a_{y,L}$ 99999

The list of the record of a course is terminated by a null driving record; that is, it is terminated by a 99999 that immediately follows 99999 of the last driving record.

Output

The test result should be reported by simply printing `OK` or `NG` for each driving record, each result in each line. No other letter is allowed in the result. Print a blank line between consecutive courses.

Sample Input

```
1

6 28 6 32 25 32 26 27 26 24 6 24 99999
2 28 2 35 30 35 30 20 2 20 99999

3 28
22.667
0 1 1 1 1 0 0 -1 0 -1 1 0 0 0 1 0 -1 0 0 -1 -1 -1 -1 0 -1 0 -1 -1 -1 1 -1 1
-1 1 -1 0 1 0 1 1 1 1 1 0 1 1 99999

5 28
22.667
0 1 -1 1 1 0 1 -1 1 -1 1 0 1 0 1 0 -1 -1 -1 0 -1 -1 -1 0 -1 1 -1 -1 -1 1
-1 0 -1 1 -1 0 1 0 1 0 1 1 1 1 1 1 99999

4 28
6.333
0 1 0 1 1 -1 -1 -1 0 -1 0 -1 0 -1 99999

3 28
20.000
0 -1 1 -1 1 0 1 1 1 1 1 0 -1 0 -1 0 -1 1 -1 1
-1 1 -1 0 -1 -1 -1 -1 -1 -1 -1 0 1 0 1 -1 1 -1 1 -1 99999

99999
```

Sample Output

```
OK
NG
NG
NG
```

Miguel A. Revilla
1999-03-05

53. 694 - The Collatz Sequence

Time limit: 3.000 seconds

The Collatz Sequence

An algorithm given by Lothar Collatz produces sequences of integers, and is described as follows:

Step 1:

Choose an arbitrary positive integer *A* as the first item in the sequence.

Step 2:

If *A* = 1 then stop.

Step 3:

If *A* is even, then replace *A* by *A* / 2 and go to step 2.

Step 4:

If *A* is odd, then replace *A* by 3 * *A* + 1 and go to step 2.

It has been shown that this algorithm will always stop (in step 2) for initial values of *A*as large as 10^9, but some values of *A* encountered in the sequence may exceed the size of an integer on many computers. In this problem we want to determine the length of the sequence that includes all values produced until either the algorithm stops (in step 2), or a value larger than some specified limit would be produced (in step 4).

Input

The input for this problem consists of multiple test cases. For each case, the input contains a single line with two positive integers, the first giving the initial value of *A* (for step 1) and the second giving *L*, the limiting value for terms in the sequence. Neither of these, *A* or *L*, is larger than 2,147,483,647 (the largest value that can be stored in a 32-bit signed integer). The initial value of *A* is always less than *L*. A line that contains two negative integers follows the last case.

Output

For each input case display the case number (sequentially numbered starting with 1), a colon, the initial value for *A*, the limiting value *L*, and the number of terms computed.

Sample Input

```
3 100
34 100
75 250
27 2147483647
101 304
101 303
-1 -1
```

Sample Output

```
Case 1: A = 3, limit = 100, number of terms = 8
Case 2: A = 34, limit = 100, number of terms = 14
Case 3: A = 75, limit = 250, number of terms = 3
Case 4: A = 27, limit = 2147483647, number of terms = 112
Case 5: A = 101, limit = 304, number of terms = 26
Case 6: A = 101, limit = 303, number of terms = 1
```

Miguel Revilla
2000-08-14

54. 729 - The Hamming Distance Problem

Time limit: 3.000 seconds

The Hamming Distance Problem

The Hamming distance between two strings of bits (binary integers) is the number of corresponding bit positions that differ. This can be found by using XOR on corresponding bits or equivalently, by adding corresponding bits (base 2) without a carry. For example, in the two bit strings that follow:

```
      A       0 1 0 0 1 0 1 0 0 0
      B       1 1 0 1 0 1 0 1 0 0
A XOR B = 1 0 0 1 1 1 1 1 0 0
```

The Hamming distance (*H*) between these 10-bit strings is 6, the number of 1's in the XOR string.

Input

Input consists of several datasets. The first line of the input contains the number of datasets, and it's followed by a blank line. Each dataset contains *N*, the length of the bit strings and *H*, the Hamming distance, on the same line. There is a blank line between test cases.

Output

For each dataset print a list of all possible bit strings of length *N* that are Hamming distance *H* from the bit string containing all 0's (origin). That is, all bit strings of length *N*with exactly *H* `1`'s printed in ascending lexicographical order.

The number of such bit strings is equal to the combinatorial symbol *C*(*N*,*H*). This is the number of possible combinations of *N*-*H* zeros and *H* ones. It is equal to

$$\frac{N!}{(N-H)!H!}$$

This number can be very large. The program should work for $1 \leq H \leq N \leq 16$.

Print a blank line between datasets.

Sample Input

```
1

4 2
```

Sample Output

```
0011
0101
0110
1001
1010
1100
```

Miguel Revilla
2000-08-31

55. 834 - Continued Fractions

Time limit: 3.000 seconds

Continued Fractions

Let $b_0, b_1, b_2, \ldots, b_n$ be integers with $b_k > 0$ for $k > 0$. The *continued fraction* of order n with coeficients $b_1, b_2, \ldots, b_n$ and the initial term b_0 is defined by the following expression

$$b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_{2+\ldots+\frac{1}{b_n}}}}$$

which can be abbreviated as $[b_0; b_1, \ldots, b_n]$.

An example of a continued fraction of order $n = 3$ is [2;3, 1, 4]. This is equivalent to

$$2 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{4}}} = \frac{43}{19}$$

Write a program that determines the expansion of a given rational number as a continued fraction. To ensure uniqueness, make $b_n > 1$.

Input

The input consists of an undetermined number of rational numbers. Each rational number is defined by two integers, numerator and denominator.

Output

For each rational number given in the input, you should output the corresponding continued fraction.

Sample Input

```
43 19
1 2
```

Sample Output

```
[2;3,1,4]
[0;2]
```

Fernando Silva, ACM-UP'2001

56. 863 - Process Scheduling

Time limit: 5.000 seconds

Process Scheduling

In batch programming, there are many tasks that need to be completed in minimal overall time. Accurate information about the total runtime and interdependencies of processes is often available to the scheduler, allowing optimal schedulings to be found. This problem considers processes that can run on different CPUs at the same time, but which are interdependent. That is to say, some processes cannot start until others have completed. Scheduling processes optimally prevents CPUs from being idle unnecessarily.

Your task is to find a way of using ***n*** processors to run ***p*** processes, minimizing the number of time slices until they are all finished. Some processes may be dependent on processes that depend on other processes, but there will be no dependency loops.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line of input will consist of two integers, the number of processors (***n***) and the number of processes (***p***). Both numbers will be positive; ***p*** < 100 and ***n*** < 21.

There will be ***p*** additional lines representing the processes. Each of these lines will contain an integer representing the number of CPU time slices required for the process and zero or more integers representing the processes that must finish before this process may start. Processes are numbered from 1 to ***p***.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

Output consists of ***n*** columns of width two, with a single space between columns. The***i***-th line corresponds to the processes running during the ***i***-th time slice. The process numbers should be right-justified. The scheduling of processes should minimize the amount of time until all of the processes have completed their tasks. Trailing spaces are acceptable, provided that they do not extend beyond the right of the last column.

Sample Input

```
1

3 5
4
3
2 4 2
2 1
1 3
```

Sample Output

```
 1  1  2
 1  1  2
 4  4  2
 3  3
 5
```

Miguel Revilla 2004-08-31

57. 894 - Juggling Trams

Time limit: 3.000 seconds

Juggling Trams

After many years of faithful service, the good city of Melbourne has decided to upgrade all of its aged green and gold trams to newer, more fashionable models. But alas, the pursuit of fashion has not been without its costs. The newer trams, whilst looking stunning, can only travel in straight lines.

But this has not deterred the Melbourne City Council. In order to deal with this problem, they have laid more tracks and built more trams, so that the city is now a vast grid of tram tracks criss-crossing throughout the city.

For simplification, consider the city to be a large grid of roads running north-south and east-west. Each road has a tram line running along it. You, the humble traveller, wish to travel south and west through this grid using the new tram system.

Trams run along each road at ***t***-minute intervals, where ***t*** is a fixed time that has been determined by the Melbourne City Council. Thus, if you stand at any street intersection, you will see a tram travelling south every ***t*** minutes, and a tram travelling west every ***t*** minutes (although the trams will not necessarily pass by at the same times). We will assume that trams run at a constant speed, and take no time to pick up or drop off passengers.

Based on your training in city traversal optimisation theory, you wish to write a computer program that will find the fastest route from your starting point to your finishing point through the grid. You can hop on or off a tram at any intersection, and you may need to wait at some intersections for a tram to come along. You may assume that hopping on or off a tram takes no time at all. So, for instance, if a southward tram passes through an intersection at the same time as a westward tram, you may hop off the southward tram and immediately hop onto the westward tram.

Input

Input will consist of a number of data sets. The first line of each data set will be of the form

t m

where ***t*** and ***m*** are both integers. ***t*** represents the number of minutes between each tram travelling down a street, as described earlier, with $1 \leq t \leq 60$ inclusive. ***m***represents the number of minutes it takes a tram to move from one intersection to the next, with ***m*** > 0. The second line of the data set will be of the form

n e

where ***n*** is the number of north-south streets and ***e*** is the number of east-west streets. Both ***n*** and ***e*** will be between 1 and 200 inclusive. North-south streets will be numbered from 1 to ***n***, where 1 is the eastmost street and ***n*** is the westmost street. East-west streets will be numbered from 1 to ***e***, where 1 is the northmost street and ***e***is the southmost street. The third line will be of the form

sx sy fx fy

and will describe the start and end points of your journey. Your journey must begin at the intersection of north-south street ***sx*** and east-west street ***sy***, and must end at the intersection of north-south street ***fx*** and east-west street ***fy***. The fourth line will contain a single integer representing the number of minutes after midnight when your journey begins.

Following this will be a line for each north-south street of the form

first k

where ***first*** is the number of minutes after midnight when the first tram leaves the eastmost intersection along that street, and ***k*** is the total number of southward trams that will travel down that street throughout the day (***k*** > 0). Note that these ***k*** trams will be spaced by intervals of ***t*** minutes. A line

first k

will then be given for each east-west street, where ***first*** is the number of minutes after midnight when the first tram leaves the northmost intersection along that street, and ***k*** is the total number of westward trams that will travel down that street throughout the day (***k*** > 0).

Input will finish with ***t*** = 0 and ***m*** = 0.

Output

Output must be one line for each case of input. If it is possible to travel from the start point to the finish point, you must output a line of the form:

`You arrive at` ***hh*****:*****mm*****.**

where ***hh*** : ***mm*** is the earliest time of day (using a 24-hour clock) at which you can arrive at your destination. Both hours and minutes must be two digits; use a leading `0' if necessary. You may assume that if a solution exists, the time of arrival will be before the next midnight.

If it is impossible to reach the destination, output the line: `Impossible.'

Sample Input

```
30 3
5 4
2 2 5 4
93
30 5
100 6
115 4
100 7
30 8
10 10
0 11
20 9
10 10
30 3
5 4
2 2 5 4
300
30 5
100 6
115 4
100 7
30 8
10 10
0 11
20 9
10 10
0 0
```

Sample Output

```
You arrive at 01:52.
Impossible.
```

Testsetter: *Joachim Wulff (Little Joey)*
Special thanks: *Ruben Spaans*

58. 896 - Board Game

Time limit: 3.000 seconds

Board Game

A surprisingly complex game can be played between two players on a simple one-dimensional board with up to 60 holes; each player has counters of one colour (indicated here by letters like W or R) which go into the holes. A player may move a counter anywhere on the board, as long as it does not land on or jump over any other counter. The object of the game is to block the other player so he or she has no allowable moves. In a game with one counter each, the first player can force a win on the first move by moving their counter next to the other counter. Whenever the second player tries to move away, the first player moves next to them. Eventually the second player will have no moves left. If the first player does not take this first move, then the second player can force a win. Convince yourself that these statements are true in this example:

```
W   R
```

With two counters each, the game is more complex. For example, in the following situation, if W moves first they can force a win by moving the leftmost counter one square to the right, or the rightmost counter one square to the left. Any other move (for example, moving one of the W counters next to an R counter) will mean than R can force a win. Try it and convince yourself this is true.

```
W   R   W R
```

Even more complex situations arise with more counters; for example, draws can occur (ie neither player can force a win) as in the following case:

```
W W   R  W R R
```

It is embarrassing that it is difficult to see how to win such a simple game. Please write a program so that I can play this safely by always knowing the winning first move, or, when there is no winning first move, whether the game is lost or can be drawn. Assume that a player will always win the game if it is in their power, and will always try to avoid defeat.

Input

Each line of input will be a game description, which is a string of up to 60 digits. Each digit represents a hole on the board: `0' for an empty hole, `1' for a counter belonging to the first player, or `2' for a counter belonging to the second player. Both players will have the same number of counters, and there will be at least two empty holes on the board. The input will be terminated by a line consisting of a single zero.

Output

For each input game description, one line of output must be produced. This line must be:

- `0' if the game cannot be won but can be drawn;
- `2' if the first player will lose whatever move is made;
- `1' if there is a winning first move. In this case, the `1' must be followed by a description of the winning move. The winning move must be given by two numbers, the first giving the hole in which the move begins, and the second giving the hole to which the piece moves. The holes are assumed to be numbered from left to right with the leftmost being number 1. If there is more than one winning move, choose the move starting from the smallest numbered hole. If there is more than one of these, choose whichever moves to the smallest numbered hole.

Sample Input

```
000200100000
000100200102
000010201002
000010200102
001020020001100002000
010122010122
0
```

Sample Output

```
1 7 5
1 4 5
1 5 4
2
1 13 15
1 2 1
```

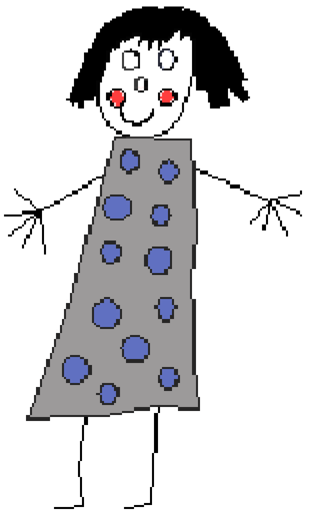
Miguel Revilla 2004-07-08

59. 913 - Joana and the Odd Numbers

Time limit: 3.000 seconds

Problem A

Joana and the Odd Numbers



Joana loves playing with odd numbers. In the other day, she started writing, in each line, an odd number of odd numbers. It looked as follows:

```
 1
 3  5  7
 9 11 13 15 17
19 21 23 25 27 29 31
...
```

On a certain line Joana wrote 55 odd numbers. Can you discover the sum of the last three numbers written in that line? Can you do this more generally for a given quantity of odd numbers?

Problem

Given the number **N** of odd numbers in a certain line, your task is to determine the sum of the last three numbers of that line.

Input

The input is a sequence of lines, one odd number **N** ($1<N<1000000000$) per line

Output

For each input line write the sum of the last three odd numbers written by Joana in that line with **N** numbers. This sum is guaranteed to be less than 2^{63}.

Sample Input

```
3
5
7
```

Sample Output

```
15
45
87
```

60. 917 - Euro 2004

Time limit: 3.000 seconds

Problem E

Euro 2004

The European Football Championship is coming! From the 12th of June to the 4th of July, Portugal will be the sports center of the world. Everyone will do their best to make this event a memorable one.

However, an important detail is missing. The rules for the classification of the league stage have changed and everyone is a little bit confused. What would be really, really nice would be a computer program to calculate this classification, given the results of the game. Then it would also be possible to watch in real time the changes in the ranking!

Here is how the classification is made:

1. For a win 3 points will be awarded; for a tie, 1 point; for a defeat, 0 points
2. For establishing the final places, the following criteria will be applied, in descending order of priority:
    - Number of points
    - Goal-Average (difference between goals scored and given)
    - Number of wins (victories)
    - Number of goals scored

So after the games are taken in account, this parameters are all calculated. Things get interesting when there is more than one team with the same number of points. In that case, a sub-league is considered. You must now imagine that only the games between the tied teams count, and see the new sub-classification. If that does not break the tie (in points) for all the teams, you must do a sub-sub-league for all the teams that are still tied, and so on. There is only one case when a sub-league should not be partitioned. That is when all the teams in that sub-league have the same number of points and obviously, the partition would give the exact same group of teams and parameters. In that cases, teams should be ranked according to the four parameters calculated for that sub-league. If the parameters are not sufficient, then the teams should be considered to be in the same place, and they should appear in alphabetical order.

Problem

Your task is to write a program that given the results of several games, calculates the classification of the teams using the sorting algorithm defined above.

Of course that knowing how good programmer you are, the organization has asked you to make a program that could calculate the classification of thousands of teams more than the ones that will be present in Euro'2004, in order to use the program in any situation they want.

Input

The first line of input contains an integer **T** which is the number of test cases that follow.

Each test case starts with a number **G** ($1 \leq G \leq 10000$) indicating the number of games to consider.

Then *G* lines follow, each one with the format `"TEAM1 TEAM2 GOALS1 GOALS2"`, giving the result of a single game (`TEAM1` scored `GOALS1` goals and `TEAM2` scored`GOALS2`). Team names are made only by lower-case letters and have a maximum length of 20.

It is not necessary that games between all the teams have been made. Of course that you should only calculate the classification based on the games that you were given. Also, some teams may play against each other more than one time.

To make the classification you should consider all the teams that played at least one game. You may assume that the number of teams is ≤ 10000

Output

The output for each test case consists of lines in the form `"PLACE TEAM"`, in ascending order of place, where PLACE indicates the place the team got and `TEAM` is the name of the team. Remember that all teams that played at least one game must appear.

Output of different test cases should be separated by a single blank line.

See the example output for a more detailed explanation of how the classification was obtained on that particular cases.

Sample Input

```
3
6
portugal grecia 4 1
espanha russia 3 1
portugal russia 3 0
espanha grecia 1 2
portugal espanha 1 3
grecia russia 7 0
```

```
6
portugal grecia 4 1
espanha russia 1 3
portugal russia 3 0
espanha grecia 1 2
portugal espanha 1 3
grecia russia 7 0
1
brasil franca 0 0
```

Sample Output

```
1 portugal
2 espanha
3 grecia
4 russia

1 portugal
2 grecia
3 russia
4 espanha

1 brasil
1 franca
```

Explanation of Sample I/O

- First sample case:

  Looking at the games, we see that `"portugal"`, `"espanha"` and `"grecia"`made 6 points, and `"russia"` made 0 points (which automatically gives them the 4th place). A tie between the first three teams is achieved. A sub-league with only that three teams is then considered but in this sub-league all the three teams have 3 points. This tied group cannot be partitioned further and then the other parameters are considered. Since in that sub-league, `"portugal"` has the best goal-average, it achieves 1st place. Then comes `"spain"` (2nd goal-average) and finally `"grecia"`. If necessary, the other parameters would have been taken in account.

- Second sample case:

  Now, `"portugal"` and `"grecia"` are tied with 6 points, and `"espanha" and "russia"` have 3 points. The sub-league between `"portugal"` and `"grecia"`unties the two teams (`"portugal"` won against `"grecia"`), and in the same way the sub-league between `"espanha" and "russia"` unties them.

- Third sample case:

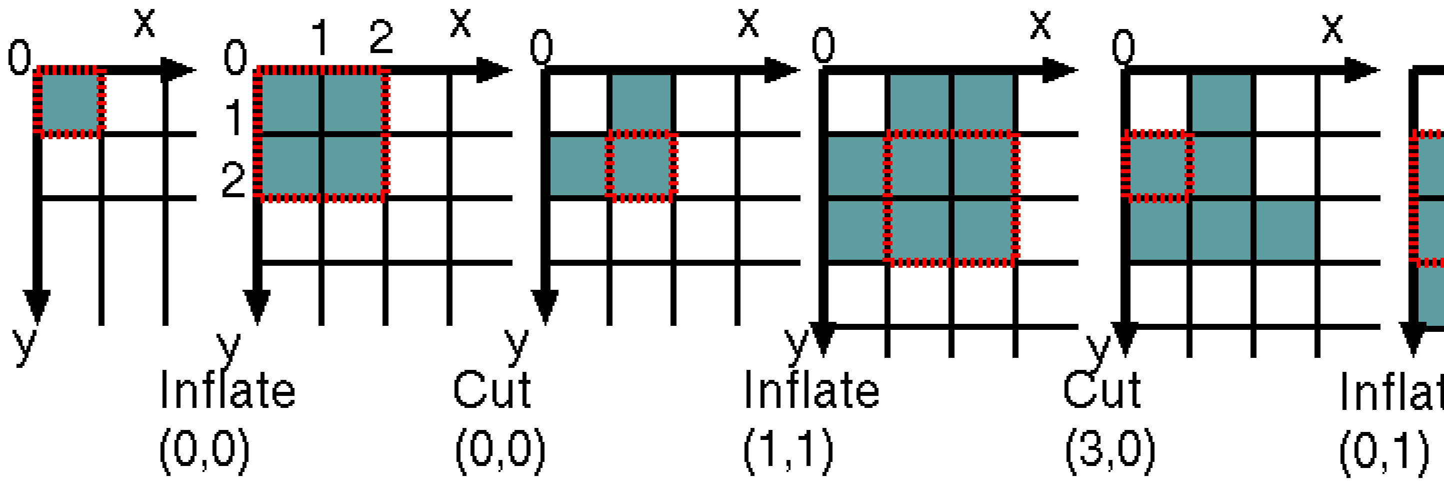
  The only two teams tied the game, so they are equal in all parameters. They are in the same place and they appear in alphabetical order.

61. 919 - Cutting Polyominoes

Time limit: 3.000 seconds

Cutting Polyominoes

A polyomino may be viewed as a set of squares connected by their sides. Its boundary is an orthogonal polygon. They are often classified by their number of squares, which is equal to their area if each square has area 1. We may represent a polymino in a grid as shown below. We are interested in polyominoes without holes and that have exactly one edge in each grid line that intersects them. Did you know that each polyomino results from a square (of area 1) by applying pairs of transformations `INFLATE/CUT`? For example, the polyomino shown on the right is obtained if one applies `INFLATE (0,0)`, `CUT (0,0)`, `INFLATE (1,1)`, `Cut (3,0)`, `INFLATE (0,1)`, `CUT (2,1)`,`INFLATE (1,2)`, `CUT (0,5)`. It easy to see that it has area `12`.



We are considering that the initial northwest corner is placed in (0,0) and that *x* and *y*grow as in the figure. `INFLATE}` *(p_i,q_i)* means ``multiplying by 4 the area of the cell" (i.e. square) whose northwest corner is in *(p_i,q_i)*. For this, we must duplicate the grid line where this cell is located and then duplicate the column where the cell was located. Obviously, we can drag down cells and then drag other cells to the right. The coordinates of the polyomino are also modified: *x* -> *x+1* iff *x* > p_i and *y* -> *y+1* iff *y* > q_i. **One cell can only be inflated if it belongs to the polyomino**.

The sequence ``INFLATE (p_i,q_i) CUT (x_i,y_i)" means that one must cut the rectangle defined by the points (p_i+1,q_i+1), (p_i+1,y_i), (x_i, y_i), (x_i, q_i+1).**Such rectangle can only be cut if it simultaneously satisfies the following conditions**:

(A) it is actually a rectangle and it is part of the polyomino;

(B) (x_i,y_i) is a vertex of the inflated polyomino and none of the other vertices of the inflated polyomino belongs to the rectangle (either to its interior or boundary);

(C) at least one of the points (x_i,q_i+1) and (p_i+1,y_i) is in an edge that contains(x_{ii}).

Problem

Your task is to write a program that computes the area of polyominoes that result from applying a sequence of transformations `INFLATE-CUT` to squares of area 1.

Input

The input is a sequence of descriptions of polyominoes's constructions, ended by 0. Each description starts with an integer $r \leq 50$, which is the number of pairs `INFLATE-CUT`, followed by *r* rows, each one with four integers p_i, q_i, x_i, y_i, that mean "`INFLATE (p`$_i$`,q`$_i$`) CUT (x`$_i$`,y`$_i$`)`". Observe that the polyomino resulting from *r* `INFLATE-CUT`'s has *2r+4* vertices.

Output

Each line of the output will have the area of the constructed polyomino or 0 if any step in the construction does not satisfy the rules just defined.

Sample Input

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

Sample Output

```
12
0
0
18
0
```

62. 951 - The Pieces of the Puzzle

Time limit: 3.000 seconds

The children on the playground are now playing with funny puzzles. They have a number of pieces and they have to make a pre-determined image. Imagine for example that you have to form the image depicted in figure 1, only with the available pieces.

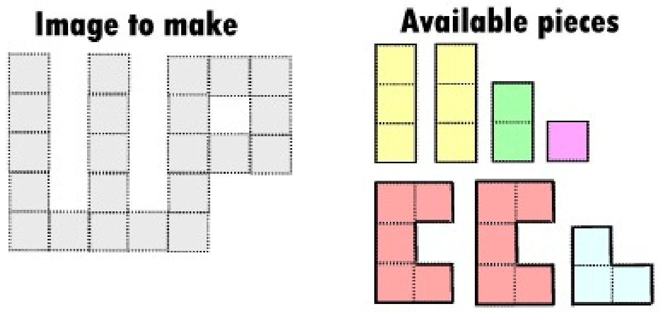


Figure 1 - The image and the pieces of the puzzle

The idea is to use all the pieces to form the image. The pieces can be rotated (by 0, 90, 180 or 260 degrees). Since the pieces are equal from both sides, they can also be flipped, and also they can be at the same time flipped and rotated. With this rules, one way to form the image with the pieces would be the one illustrated in figure 2.

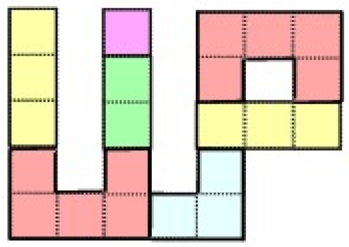


Figure 2 - Combining the pieces to form the puzzle

Can you help the children in how to solve every puzzle they have in hands?

The Problem

Given a set of pieces and an image, you have to discover if it is possible to combine the pieces in order to form the image. An image is always a connected set of squares (meaning that there are no disconnected parts of the image). At the end, there can be no pieces left.

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

The first line of the input will contain two integers **H** and **W**, which respectively indicate the height and the width of the minimal square than can enclose the image to form ($1 \leq H,W \leq 20$). Than come exactly ***H*** lines describing the image itself, each of these lines exactly with ***W*** chars. A `'#'` means a square that belongs to the image and a `'.'`means one that does not belong.

After that comes a single line with an integer **P** indicating the number of pieces available ($1 \leq P \leq 400$), followed by ***P*** sets of lines describing the pieces, in a similar way of how the image is described. Each piece starts by having a single line indicating $\mathbf{PH_i}$ and$\mathbf{PW_i}$, the height and width of the minimal square enclosing the piece ($1 \leq \boldsymbol{PH_i}, \boldsymbol{PW_i} \leq 20$), followed by $\boldsymbol{PH_i}$ lines with $\boldsymbol{PW_i}$ chars with the piece, where `'#'` means a square that belongs to the piece and a `'.'` means one that does not belong. Note that the pieces can come in any order.

Output

For each test case, you must output a single line with the word `Yes` if the image can be made with the pieces or `No` if that is not possible. Remember that all the pieces must be used and that the pieces can be rotated, flipped or suffer both of these transformations.

Sample Input

```
5 7
X.X.XXX
X.X.X.X
X.X.XXX
X.X.X..
XXXXX..
7
3 1
X
X
X
3 1
X
X
X
2 1
X
X
1 1
X
3 2
XX
X.
XX
3 2
XX
X.
XX
2 2
X.
XX

3 3
.X.
XXX
.X.
2
2 2
.X
XX
2 1
X
X

3 2
.X
.X
XX
1
3 2
X.
X.
XX
```

Sample Output

```
Yes
No
Yes
```

63. 961 - Ambiguous or Incomplete Inductive Definitions

Time limit: 3.000 seconds

Ambiguous or Incomplete Inductive Definitions

An important notion in Computer Science is the syntactical concept of terms. Examples of terms can be:

- `a`
- `b`
- `f(a,b)`
- `s(a)`
- `g(f(a,b),b,s(a))`

Sets of terms are usually defined by induction. In such a schema, a set of terms is seen in a constructive way: each element of an inductively defined set is either constructed from simpler elements of the set or a basic element. For instance, `a` is a basic element,`g(f(a,b),b,s(a))` is constructed from `f(a,b)`, `b` and from `s(a)` using the constructor `g`. In the same vein, `f(a,b)` is constructed from `a` and from `b` using the constructor `f`.

More formally, an inductive definition of a set ***T*** of terms is composed by a non-empty set ***B*** of basic elements and a set ***K*** of constructors. Then, we say that ***T*** is the smallest set ***X*** containing ***B*** (i.e. ***B*** $\subseteq$ ***X***) and the elements that respect the following rule:

let be ***f*** $\in$ ***K*** with an arity of ***n***, and let be ***n*** elements of ***X*** (say, $\boldsymbol{a}_1 \ldots \boldsymbol{a}_n$) then ***f*** $(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_n)$ must be in ***X***

Let $\boldsymbol{h}_B : \boldsymbol{B} \longrightarrow \boldsymbol{N}$ be a total function that maps every symbol of ***B*** to a positive integer, and $\boldsymbol{h}_K$ be a total function over ***K*** that maps every ***n*** -ary constructor to an ***n***-ary function over positive integers.

In such a setting, we define the notion of natural interpretation ***h*** as a function from ***T*** to***N*** that maps every term ***t*** $\in$ ***T*** to a positive integer in the following way:

$$\begin{cases} h(a) & = & h_B(a) \quad \text{if} \quad a \in B \\ h(f(a_1 \ldots a_n)) & = & h_K(f)(h(a_1) \ldots h(a_n)) \end{cases}$$

We say that an inductive definition ***T*** paired with a natural interpretation ***h*** is ambiguouswhen there exist two terms $\boldsymbol{t}_1, \boldsymbol{t}_2 \in$ ***T*** such that $\boldsymbol{h}(\boldsymbol{t}_1) = \boldsymbol{h}(\boldsymbol{t}_2)$. We also say that (***T***, ***h***)is incomplete when there exists a positive integer ***n*** such that there is no term ***t*** that verifies $\boldsymbol{h}(\boldsymbol{t}) = \boldsymbol{n}$. Finally we say that (***T***, ***h***) is regular if it is neither incomplete nor ambiguous.

Given an inductive definition of a set ***T*** of term and a natural interpretation ***h***, your task consists in qualifying if (***T***, ***h***) is ambiguous, incomplete, both or regular.

In the context of this problem, we will only consider simple interpretations. As a consequence, elements in h_K are simple functions defined by the following grammar:

f : : = (*f* + *f*) | (*f***f*) | var_{id} | *N*

For a ***p***-ary function, the only valid var_{id} are `x1`, `x2` ... `xp`, `x1` for the first argument, `x2`for the second argument, and so on. Consider that every component in the definition of a function is separated from the other by a single space. For instance the successorfunction is described by `( x1 + 1 )`.

In order to simplify the problem, you will not have to consider the whole set of natural numbers. You only will have to consider the set {***N***..***M***} with $0 \leq N < M \leq 30000$, both provided by the input data. Consider also that each constructor have at least one parameter and at most 5 parameters.

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

The input consists in the following lines:

- the first line contains ***N*** and ***M***, in this order and separated by a single space;
- the second line contains a single integer ***n*** ($1 \leq n$) that represents the number of elements of ***B***;
- the next line contains a single integer ***m*** ($0 \leq m$) that is the number of constructors;
- the following ***m*** lines introduce the arity of the ***m*** constructors. Thus, each line contains a single integer;
- the next ***n*** lines define the function h_B. The first of these lines contains an integer***x*** (= $h_B(a)$) related to the first basic element ***a*** (remember, $N \leq x \leq M$). And so on;
- the last ***m*** lines define h_K. Each line is then the description of a function that respects the grammar exposed above.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The output is organized following one of these four situations:

Case (*T*, *h*) is regular:

a single line with the word REGULAR.

Case (*T*, *h*) is incomplete:

a single line with the word `INCOMPLETE`, a single space followed by a integer that is the smallest value that cause the incompleteness of (***T***, ***h***).

Case (*T*, *h*) is ambiguous:

a single line with the word `AMBIGUOUS`, a single space followed by a integer that is the smallest value that turns (***T***, ***h***) ambiguous.

Case (*T*, *h*) is both incomplete and ambiguous:

the output is, in this case, two lines long. The first line reports the incompleteness along the lines of the second case. The second line reports the ambiguity of (***T***,***h***) in the same way as the third case.

Sample Input

```
0 30000
1
1
1
0
( x1 + 1 )

0 30000
1
1
1
1
( x1 + 1 )

0 30000
1
2
1
2
1
( x1 + 1 )
( x1 + x2 )
```

Sample Output

```
REGULAR

INCOMPLETE 0

INCOMPLETE 0
AMBIGUOUS 2
```

MIUP'2006

64. 964 - Custom Language

Time limit: 3.000 seconds

CUSTOM LANGUAGE

Problem

We would like to use a custom language in the TIUP contest for writing our solutions, but we are not allowed to. However, as we like it so much, we want you to develop an interpreter for it! Our language is very simple, it works over a stack of integers and contains the following set of instructions:

`PUSH v` : puts value `v` on the top of the stack;

`POP x` : removes the value at the top of the stack and puts it on variable `x`;

`DUP` : duplicates the top of the stack, i.e, repeats the value at the top by pushing it again;

`ADD` : adds the two values at the top of the stack;

`SUB` : subtracts the two values at the top of the stack;

`MUL` : multiplies the two values at the top of the stack;

`DIV` : divides the two values at the top of the stack;

`READ` : reads a value from input and puts it on the top of the stack;

`WRITE` : writes the value at the top of the stack on output followed by a `\n`;

`JUMP v` : jumps to instruction `v`;

`JUMPPOS v` : jumps to instruction `v`, if the top of the stack is greater than 0;

`JUMPZERO v` : jumps to instruction `v`, if the top of the stack is 0;

A program is a list of instructions, one per line. An instruction is identified by the line number `i`, where $1 \le i \le |L|$. Whenever we reach an instruction not defined, the program ends. For noncommutative operations, the top of the stack is the first argument. A variable `s` is a string in $\{a, \ldots, z\}.\{1, \ldots, 9, a, \ldots, z\}^*$ such that $|s| \le 100$. In the above, v can be a constant or a variable. All arithmetic operations remove their arguments from the top of the stack and put their output on the top of the stack. Finally, when any instruction reads the top of the stack it removes it.

Input

Several programs, each consisting of a code section and a data section. A code section consists of a list of instructions, one per line, and is terminated by a line with the symbol #. This is followed by the a data section consisting of a set of integers, one per line, which are the inputs for the program. A data section is also terminated by a line with the symbol #. Each program has one code section and one data section; a data section can be empty.

Output

For each program the output obtained by the program for the given input (if any) orABORTED if something wrong happens. The output of each program should be terminiated by a line with the symbol #.

Sample Input

```
READ
POP num
PUSH 2
POP x
PUSH num
PUSH 2
SUB
JUMPPOS 31
PUSH num
PUSH x
SUB
DUP
JUMPZERO 28
JUMPPOS 28
PUSH x
PUSH num
DIV
PUSH x
MUL
PUSH num
SUB
JUMPZERO 31
PUSH x
PUSH 1
ADD
POP x
JUMP 9
PUSH 1
WRITE
JUMP 33
PUSH 0
WRITE
#
7
#
READ
READ
DIV
WRITE
JUMP 1
#
5
25
25
5
```

```
0
1
7
8
#
PUSH 0
POP m
PUSH 10
DUP
PUSH m
ADD
DUP
POP m
WRITE
POP temp
PUSH 1
PUSH temp
SUB
DUP
JUMPPOS 4
#
#
PUSH undefined
WRITE
#
#
READ
POP defined
PUSH defined
WRITE
#
-14
#
```

Sample Output

```
1
#
ABORTED
#
10
19
27
34
40
45
49
52
54
55
#
ABORTED
#
-14
#
```

65. 977 - Old West Rumours

Time limit: 3.000 seconds

Problem F - Old West Rumours

Never have rumours been so fast as in the internet era. Although not as fast, in the old west, they still runned pretty darn fast. As soon as a rumour started in one town it would quickly spread to its neighbours in a matter of hours. Besides that rumours could easily destroy many kinds of deals, both legal and illegal.

This was the context that allowed Lightning Billy to make his living. Billy had a business where he sold to his customers the usefull service of stopping rumours. When a customer came to Billy he would give Billy the details of the rumour being spread. This allowed Billy to estimate how much time he would need to stop that rumour in any given town. With this information and knowing all the possible routes the rumour could take, Billy could estimate in how many towns he could kill the rumour before it reached them (See Figure 1).

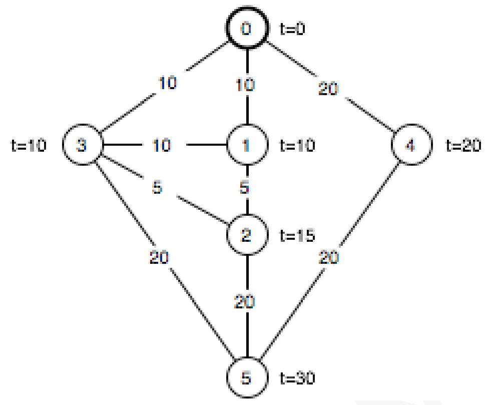


Figure 1 - Towns, routes and rumour distances

The Problem

Your problem will be to design an algorithm that given a set of towns and routes between them will calculate how many cities Billy can reach in time, knowing that Billy could travel twice as fast as any rumour (he wasn't called Ligthning Billy for nothing). The time needed by Billy to travel through any route will always be rounded down (yes, he really was that fast). Billy will be able to kill the rumour only if he manages to reach a town at least at the same time the rumour gets there.

Input

The input file contains several test cases, each of them as describes below.

The first line of the input will contain the number of towns in the area (*1<=nt<=50*), and the number of routes (*0<=nr<=1000*), between them. Then, a single line containing *nt* values with the time needed by Billy to stop the rumour in each town (*0<=s<=10*), followed by *nr* lines containing routes. Each route will be composed by the starting city, the ending city and the time needed for the rumour to spread along that path (*1<=t<=200*). Note that rumours run both ways between the cities of the route at the same speed, and so does Billy (actually, at half the speed of the rumours, as described above).

Output

For each test case, output a single line containing the number of towns where Billy can stop the rumour. Billy always starts from the first town in the input file. Towns that are not reached by the rumour should not be counted as they are not saved by Billy.

Example Input

```
6 9
1 2 3 4 5 6
0 1 10
0 3 10
0 4 20
1 3 10
1 2 5
2 3 5
2 5 20
3 5 20
4 5 20
```

Example Output

```
4
```

Output explanation: Billy loses 1 time unit to solve the rumour at town 0. He then travels to town 1 taking 5 time units (10/2) and arriving there at time unit 6. He loses 2 time units there solving the rumour and then travels to town 2 taking 2 time units and arriving at time unit 10. He loses 3 time units at this town, departing at time unit 13 to town 5 where he arrives at time unit 23 with plenty of time to solve the rumour in that town (See Figure 2). No other towns can be kept safe from the rumour.

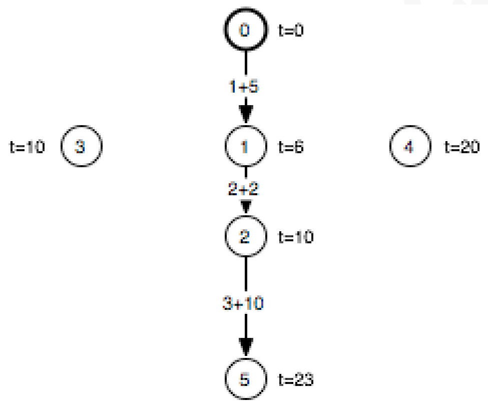


Figure 2 - Output Explanation

66. 973 - The Guessing Game

Time limit: 3.000 seconds

Problem B - The Guessing Game

Chaos is installed in Sesame Street! Elmo, the furry red creature, has created a new game and everyone wants to play it. Oscar the Grouch, who loves trash, says he is the one who plays the game better. Bert and Ernie say that they rock. The monster Herry disagrees, and simply says that no one can play the game. Grover thinks that he knows the secret to the game. The Cookie Monster cannot believe that someone would rather prefer to play the game than to eat cookies. Everyone is screaming against each other!

Meanwhile, Kermit the Frog is observing everyone and secretly preparing the strategy that will make him the most respected muppet: learning how to master Elmo's new game. But what game is this?

Elmo has a box filled with delicious cookies. Inside, the box has a grid shape, but Elmo does not show the contents of the box to anyone. Instead, he just says how many cookies are in each row, in each column and in each diagonal. For example, imagine the following cookie box (where 'X' stands for a cookie, and '.' for an empty spot). Figure 1 shows the numbers of cookies in each row and in each column.

```
. X . . X X . . 3
. X . . X . . . 2
. . . X . X X X 4
. X . . . . . . 1

0 3 0 1 2 2 1 1
```

Figure 1 - Cookies in each row and column

In what respects do diagonals, everyone in Sesame Street knows what to do, but since you may not know their standards, they number their diagonals starting from the bottom left and ending in the upper right, like it is depicted in figure 2.

```
456789
345678
234567
123456
```

Figure 2 - Diagonals in Sesame Street

So given this order of the diagonals, the number of cookies in the diagonals for the box of figure 1 would be given by `0 1 0 1 2 0 2 2 2 0 0`.

Given the number of cookies in each column, row and diagonal, Elmo's guessing game is to discover the contents of the box. If someone can guess the positions of the box, Elmo will give this person all the cookies!

You must help Kermit the Frog in mastering this game...

The Problem

Given the number of cookies in each column, row and Diagonal of Elmo's box, you must discover the positions of the cookies inside the box. If there is more than one possible solution to the contents of the box, you should discover how many possible arrangements are there that fulfill the conditions and would give the same number of cookies in rows, columns and diagonals.

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

The first line of input contains two integers, **R** and **C**, representing respectively the number of rows and columns of the cookie box (*1 &le R,C ≤ 10*).

Then comes a line with exactly *R* integers separated by a space, indicating the number of cookies in each row, from the top to the bottom.

After that there is line with exactly *C* integers separated by a space, indicating the number of cookies in each column, from the left to the right.

Finally comes a line with exactly *R+C-1* integers separated by a space, indicating the number of cookies in each diagonal, from the bottom right to the top left.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

If there is a single possible solution (arrangement of cookies) for the given input, then you should output the contents of the box, by printing *R* lines, each one with *C* chars, with 'X' representing cookies and '.' representing empty spots.

If there is more than one possible solution, simply print a line with a single integer, indicating the number of possible arrangements of cookies in the box that would give the numbers of the input for quantities in rows, columns and diagonals. You can be sure that there will never be more than 10000 valid arrangements for the test cases in the input.

Sample Input

```
4 8
3 2 4 1
0 3 0 1 2 2 1 1
0 1 0 1 2 0 2 2 2 0 0

5 5
1 1 1 1 0
1 1 1 0 1
0 0 0 1 1 2 0 0 0
```

Sample Output

```
.X..XX..
.X..X...
...X.XXX
.X......

2
```

67. 984 - Finding Haplotypes

Time limit: 3.000 seconds

Universidade da Beira Interior21-10-2006

Problem F

FINDING HAPLOTYPES

Problem

Everybody inherits two copies of each chromossome (except for the X and Y chromossomes), one for each parent. In some positions of the DNA, (about 0.1% of them) there are variations in the base that is present. These are called Single Nucleotide Polymorfisms (SNPs). In most cases, there are only two possible values in each position, that will be denoted by 0 and 1.

When DNA is sequenced, stretches from both copies are pasted together, and is impossible, with present technology, to identify from which chromossome each copy came. Computers are called to help. Your job is to figure out the haplotypes (original chromossomes) that generated a given set of genotypes.

A genotype is given by a sequence of 0s, 1s and 2s. A 0 means that the genotype is homozygotic at that position and the SNP has value 0; 1 means that the genotype is homozygotic at that position and the SNP has value 1; and a 2 means the genotype is heterozygotic at that position and that the SNP has value 0 in one haplotype and 1 in the other haplotype. An haplotype is a sequence of 0s and 1s.

Given a set of genotypes, the objective is to find a minimum sized set of haplotypes that can be combined in pairs to obtain the given genotypes. For example, given the three genotypes 02120, 20120 and 22110, it is enough to have three haplotpes to explain them: 00100, 10110 and 01110. Haplotype 00100 combines with 10110 to give genotype 20120, and so on.

Given a set of genotypes, you should print the minimum number of haplotypes that is necessary to explain it.

Input

The input file contains several test cases, each of them as describes below. The input file contains two numbers in the first line, that indicate the number of genotypes that follow (*N*) and the length of each genotype (*L*). Is is followed by *N* lines, each with *L*characters that describe each genotype. *N* will never be larger than 12 and *L* will never be larger than 10.

Output

For each test case, the output should consist of one integer, that represents the minimum number of haplotypes necessary to explain the given genotypes, on a line by itself.

Sample Input

```
3 5
02120
20120
22110
```

Sample Output

```
3
```

Maratona Inter-Universitária de Programação 2006MIUP'2006

Author: Arlindo Oliveira (Instituto Superior T�cnico)