ACM International Collegiate Programming Contest 2000/2001

Mid-Central European Regional Contest

Albert-Ludwigs University, Freiburg, Germany

November 19th, 2000

This problem set should contain eight (8) problems on thirteen (13) numbered pages. Please inform a runner immediately if something is missing from your problem set.

ACM International Collegiate Programming Contest 2000/2001
Mid-Central European Regional Contest

Problem A

Atlantis

Source: `atlantis.(c|cc|pas|java)`
Input: `atlantis.in`

There are several ancient Greek texts that contain descriptions of the fabled island Atlantis. Some of these texts even include maps of parts of the island. But unfortunately, these maps describe different regions of Atlantis. Your friend Bill has to know the total area for which maps exist. You (unwisely) volunteered to write a program that calculates this quantity.

Input

The input file consists of several test cases. Each test case starts with a line containing a single integer n ($1 \leq n \leq 100$) of available maps. The n following lines describe one map each. Each of these lines contains four numbers x_1, y_1, x_2, y_2 ($0 \leq x_1 < x_2 \leq 100\,000, 0 \leq y_1 < y_2 \leq 100\,000$), not necessarily integers. The values (x_1, y_1) and (x_2, y_2) are the coordinates of the top-left resp. bottom-right corner of the mapped area.

The input file is terminated by a line containing a single 0. Don't process it[1].

Output

For each test case, your program should output one section. The first line of each section must be "`Test case #`*k*", where k is the number of the test case (starting with 1). The second one must be "`Total explored area:` *a*", where a is the total explored area (i.e. the area of the union of all rectangles in this test case), printed exact to two digits to the right of the decimal point.

Output a blank line after each test case.

Sample Input

```
2
10 10 20 20
15 15 25 25.5
0
```

Sample Output

```
Test case #1
Total explored area: 180.00
```

[1] We warned you!

ACM International Collegiate Programming Contest 2000/2001
Mid-Central European Regional Contest

Problem B

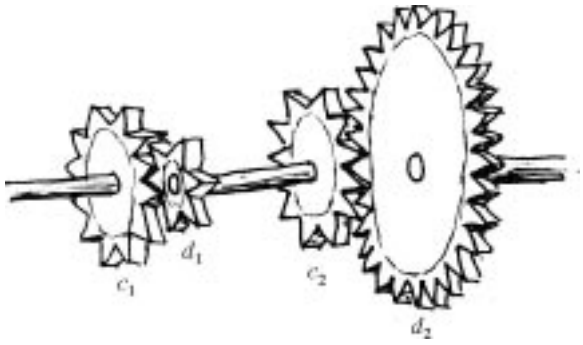
Cog-Wheels

Source: `cogwheels.(c|cc|pas|java)`
Input: `cogwheels.in`

Your little sister got a new mechanical building kit, which includes many cog-wheels of different sizes. She started building gears with different ratios, but soon she noticed that there were some ratios which were quite difficult to realize, and some others she couldn't realize at all. Since the most intelligent being in your family is your computer (as you have always been claiming), now it has to figure out which ratios are achievable.

There you are! That's the punishment for bragging about your computer. Now, you must write a program that will do the job: your sister tells you the sizes of the cog-wheels (the numbers of cogs they have) in her kit. Then, she tells you the ratios of the gears she would like to build. Your computer has to decide whether this is possible and, if so, determine how to connect the wheels to obtain the ratio.

Here is an example: let's assume there are cog-wheels with 6, 12, and 30 cogs. Your sister wants to realize a gear of ratio 4 : 5. One possible solution is the following:



The picture shows a complete gear of ratio 4 : 5. Four wheels are used: c_1 with 12 cogs, d_1 with 6 cogs, c_2 with 12 cogs, and d_2 with 30 cogs. Note that d_1 and c_2 share one axis. In this configuration, if c_1 turns once, d_2 will make $\frac{4}{5}$ of a rotation.

On the other hand, no gear of ratio 1 : 6 can be realized using the cog-wheels your sister has.

The picture above can be written as $12:6, 12:30$. Each transmission is written as $c:d$, where c and d denote the number of cogs of the two wheels. A list of transitions $c_1:d_1, c_2:d_2, \ldots, c_m:d_m$ means that the second wheel of each transition is on the same axis as the first one of the next transition (d_i and c_{i+1} share one axis for $1 \leq i < m$). For those of you who are not good at mechanics: the ratio realized by this gear is

$$\prod_{i=1}^{m} \frac{c_i}{d_i}.$$

Input

The input file contains the descriptions of several sets of cog-wheels, each one followed by a list of ratios to be realized.

A set of cog-wheels is described by one line starting with the number n of sizes of cog-wheels ($1 \leq n \leq 20$). The rest of the line will consist of n numbers $a_1, \ldots, a_n$, the numbers of cogs on the wheels. There will always be at least 5 and at most 100 cogs per wheel. You may assume that your sister has an infinite supply of wheels of each size.

In your sister's building kit (and thus, in the input file), the number of cogs on every wheel is divisible by the number of cogs on the smallest wheel in the kit.

The line describing the set of cog-wheels is followed by a list of ratios to be realized. Each ratio to be realized is given by one line containing two numbers a_j and b_j ($1 \leq a_j, b_j \leq 10000, a_j \neq b_j$), meaning $a_j : b_j$. The line "`0 0`" marks the end of that list.

At the end of the input file, there will be a line containing only a zero (instead of the number of cog-wheels of the next set).

Output

Output one section for each set of cog-wheels. The sections should start with the line "`Set #`k" where k is the number of the set.

Then, output the results for the ratios from the test set. Output exactly one line for each ratio. Each of these lines should start with "`Ratio` a_j`:`b_j`:`", followed either by "`Impossible`" if the ratio cannot be realized with the given set of cog-wheels, or a description of a gear which realizes the ratio. This description has to be in the notation described above: a blank-separated list of transitions, each transition having the form "c_i`:`d_i".

The gear does not need to be the smallest possible; we guarantee that if there is a solution then there is a solution using at most 10 000 transitions. Any solution with at most that many transitions is an acceptable answer.

Every section should be followed by a blank line.

Sample Input

```
3 6 12 30
4 5
1 6
0 0
0
```

Sample Output

```
Set #1
Ratio 4:5: 12:30 12:6
Ratio 1:6: Impossible
```

Problem C

Erdös Numbers

Source: `erdos.(c|cc|pas|java)`
Input: `erdos.in`

The Hungarian Paul Erdös (1913–1996, pronounced as "Ar-dish") was not only one of the strangest mathematicians of the 20th century, he was also among the most famous ones. He kept on publishing widely circulated papers up to a very high age, and every mathematician having the honor of being a co-author to Erdös is well respected.

Not everybody got a chance to co-author a paper with Erdös, so many people were content if they managed to publish a paper with somebody who had published a paper with Erdös. This gave rise to the so-called *Erdös numbers*. An author who has jointly published with Erdös had Erdös number 1. An author who had not published with Erdös but with somebody with Erdös number 1 obtained Erdös number 2, and so on. Today, nearly everybody wants to know what Erdös number he or she has. Your task is to write a program that computes Erdös numbers for a given set of scientists.

Input

The input file contains a sequence of scenarios, each scenario consisting of a paper database and a list of names. A scenario begins with the line "p n", where p and n are natural numbers with $1 \leq p \leq 32000, 1 \leq n \leq 3000$. Following this line are p lines containing descriptions of papers (this is the paper database). A paper is described by a line of the following form:

LastName1, *FirstName1*, *LastName2*, *Firstname2*, ...: *TitleOfThePaper*

The names and the title may contain any ASCII characters between 32 and 126 except commas and colons. There will always be exactly one space character following each comma. The first name may be abbreviated, but the same name will always be written in the same way. In particular, Erdös' name is always written as "`Erdos, P.`". (Umlauts like 'ö','ä',... are simply written as 'o','a', ...)

Example:

```
Smith, M.N., Martin, G., Erdos, P.:  Newtonian forms of prime factors
matrices.
```

After the p papers follow n lines each containing exactly one name in the same format as in the paper database.

The line '`0 0`' terminates the input.

No name will consist of more than 40 characters. No line in the input file contains more than 250 characters. In each scenario there will be at most 10 000 different authors.

Output

For every scenario first print the number of the scenario in the format shown in the sample output. Then print for every author name in the list of names their Erdös number based on the papers in the paper database of the scenario. The authors should be output in the order given in the input file. Authors that do not have any relation to Erdös via the papers have Erdös number `infinity`. Adhere to the format shown in the sample output.

Sample Input

```
2 2
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factors matrices.
Gardner, M., Martin, G.: Commuting Names
Smith, M.N.
Gardner, M.
0 0
```

Sample Output

```
Database #1
Smith, M.N.: 1
Gardner, M.: 2
```

Problem D

Number Game

Source: `numbergame.(c|cc|pas|java)`
Input: `numbergame.in`

Christine and Matt are playing an exciting game they just invented: the Number Game. The rules of this game are as follows.

The players take turns choosing integers greater than 1. First, Christine chooses a number, then Matt chooses a number, then Christine again, and so on. The following rules restrict how new numbers may be chosen by the two players:

- A number which has already been selected by Christine or Matt, or a multiple of such a number, cannot be chosen.
- A sum of such multiples cannot be chosen, either.

If a player cannot choose any new number according to these rules, then that player loses the game.

Here is an example: Christine starts by choosing 4. This prevents Matt from choosing 4, 8, 12, etc. Let's assume that his move is 3. Now the numbers 3, 6, 9, etc. are excluded, too; furthermore, numbers like: $7 = 3+4, 10 = 2\cdot 3+4, 11 = 3+2\cdot 4, 13 = 3\cdot 3+4, \ldots$ are also not available. So, in fact, the only numbers left are 2 and 5. Christine now selects 2. Since $5 = 2+3$ is now forbidden, she wins because there is no number left for Matt to choose.

Your task is to write a program which will help play (and win!) the Number Game. Of course, there might be an infinite number of choices for a player, so it may not be easy to find the best move among these possibilities. But after playing for some time, the number of remaining choices becomes finite, and that is the point where your program can help. Given a game position (a list of numbers which are not yet forbidden), your program should output all *winning moves*.

A winning move is a move by which the player who is about to move can force a win, no matter what the other player will do afterwards. More formally, a winning move can be defined as follows.

- A winning move is a move after which the game position is a losing position.
- A winning position is a position in which a winning move exists. A losing position is a position in which no winning move exists.
- In particular, the position in which all numbers are forbidden is a losing position. (This makes sense since the player who would have to move in that case loses the game.)

Input

The input file consists of several test cases. Each test case is given by exactly one line describing one position.

Each line will start with a number n ($1 \le n \le 20$), the number of integers which are still available. The remainder of this line contains the list of these numbers $a_1, \ldots, a_n$ ($2 \le a_i \le 20$).

The positions described in this way will always be positions which can really occur in the actual Number Game. For example, if 3 is not in the list of allowed numbers, 6 is not in the list, either.

At the end of the input file, there will be a line containing only a zero (instead of n); this line should not be processed.

Output

For each test case, your program should output "`Test case #`*m*", where *m* is the number of the test case (starting with 1). Follow this by either "`There's no winning move.`" if this is true for the position described in the input file, or "`The winning moves are:` w_1 w_2 ... w_k" where the w_i are all winning moves in this position, satisfying $w_i < w_{i+1}$ for $1 \leq i < k$. After this line, output a blank line.

Sample Input

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

Sample Output

```
Test Case #1
The winning moves are: 2

Test Case #2
There's no winning move.

Test Case #3
The winning moves are: 4 5 6
```

Problem E

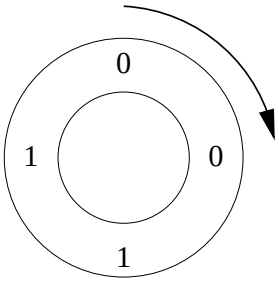
Ouroboros Snake

Source: `ouroboros.(c|cc|pas|java)`
Input: `ouroboros.in`

Ouroboros is a mythical snake from ancient Egypt. It has its tail in its mouth and continously devours itself.

The Ouroboros numbers are binary numbers of 2^n bits that have the property of "generating" the whole set of numbers from 0 to $2^n - 1$. The generation works as follows: given an Ouroboros number, we place its 2^n bits wrapped in a circle. Then, we can take 2^n groups of n bits starting each time with the next bit in the circle. Such circles are called *Ouroboros circles* for the number n. We will work only with the smallest Ouroboros number for each n.

Example: for $n = 2$, there are only four Ouroboros numbers. These are $0011, 0110, 1100,$ and 1001. In this case, the smallest one is 0011. Here is the Ouroboros circle for 0011:



k	`00110011...`	$o(2,k)$
0	`00`	0
1	`01`	1
2	`11`	3
3	`10`	2

The table describes the function $o(n,k)$ which calculates the k-th number in the Ouroboros circle of the smallest Ouroboros number of size n. This function is what your program should compute.

Input

The input consists of several test cases. For each test case, there will be a line containing two integers n and k ($1 \leq n \leq 15$; $0 \leq k < 2^n$). The end of the input file is indicated by a line containing two zeros. Don't process that line.

Output

For each test case, output $o(n,k)$ on a line by itself.

Sample Input

```
2 0
2 1
2 2
2 3
0 0
```

Sample Output

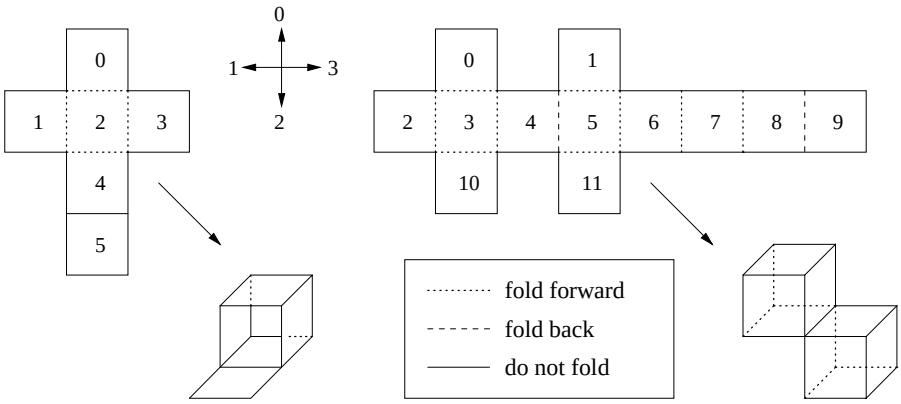
```
0
1
3
2
```

Problem F

Fold-up Patterns

Source: `patterns.(c|cc|pas|java)`
Input: `patterns.in`

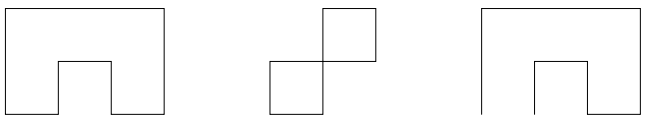
Fold-up patterns for solids like cubes or octahedrons can be found in many books on geometry, but without actually folding them it is hard to tell whether the constructions really work. In this problem, we will consider a special class of such patterns.



Given a fold-up pattern built from unit squares in the plane, together with a description along what edges it should be folded in what direction, decide whether it will result in a closed surface of a solid in three dimensions. If it does, find the volume of the solid.

More precisely, the pattern consists of a connected set of unit squares in the plane. For any edge between connected sides you are told whether to fold forward, backward (always at a right angle), or not at all along that edge. If an edge between two adjacent squares in the pattern is not mentioned in the input, you may assume that the squares are not connected and can be ripped apart when folding. However, connected edges must always be folded according to the description.

For our purposes a closed surface is one where every square in the pattern separates the inside from the outside. When folded, the squares of the pattern lie on a rectangular, 3-dimensional grid, and each separates a cell (cubes of side length one unit) on the inside from one on the outside. For every cell it must be clear whether it is inside or outside. The following sketch illustrates this rule in two dimensions.



Two closed surfaces Not a closed surface

Note that even the second pattern above satisfies our definition of a closed surface, but the interior is not connected.

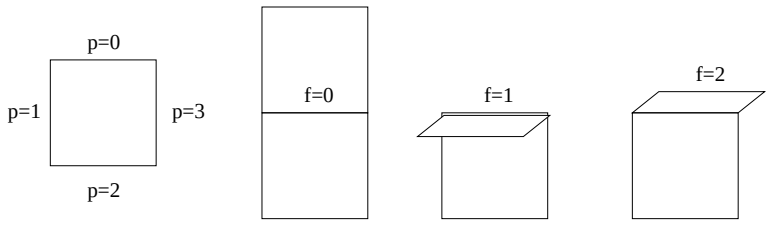
Two different squares may not occupy exactly the same position in space, though they may (and will for a closed surface) touch at edges and vertices. Make sure that the pattern does not interpenetrate itself through connected edges. Apart from that, do not worry about the process of folding, e.g. what edges are folded first or whether part of the structure is in the way for the rest.

Input

The input file consists of several test cases.

For each test case, the first line contains two integers n and e. These are the number n ($1 \leq n \leq 200$) of squares in the pattern and the number e ($0 \leq e \leq 300$) of edges. Squares are labelled by the integers 0 to $n-1$. The following e lines describe one edge each using the four numbers s_1, s_2, p, f:

- The two numbers s_1 and s_2 (with $0 \leq s_1 < s_2 < n$) of the squares that are joined by the edge.
- The position p of the square s_2 with respect to the square s_1 in the pattern. Here $p = 0, 1, 2, 3$ mean above, to the left, below, or to the right of s_1, respectively (see sketch below).
- The number $f = 0, 1, 2$ tells you to fold along the edge either not at all, forward, or back, respectively (see sketch).



You can also assume that the pattern is connected and can be drawn in the plane without overlapping.

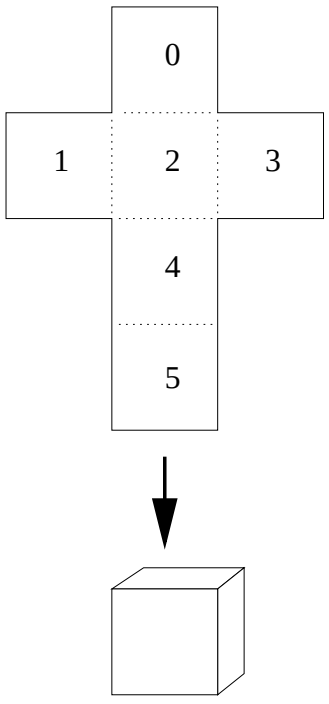
At the end of the input file, there will be a line containing two zeros (instead of n and e). Do not process that line.

Output

For each scenario print "`Test case #k:`", where k is the number of the test case (starting from 1). Then, on the same line, print either "`not a closed surface`" if the pattern does not form a closed surface or "`closed surface, volume=`" and the volume as an integer if it does.

Sample Input

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



Sample Output

```
Test case #1: closed surface, volume=1
Test case #2: not a closed surface
```

Problem G

Railroad

Source: `railroad.(c|cc|pas|java)`
Input: `railroad.in`

It is Friday evening and Jill hates two things which are common to all trains:

1. They are always late.
2. The posted schedule is always wrong.

Nevertheless, tomorrow in the early morning hours Jill will have to travel from Tuttlingen to Freiburg in order to get to the Regional Programming Contest. Since she is afraid of arriving too late and being excluded from the contest, she is looking for the train which gets her to Freiburg as early as possible. However, she dislikes getting to the station too early, so if there are several schedules with the same arrival time, she will choose the one with the latest departure time.

Jill asks you to help her with her problem, so that she can sleep a bit longer tomorrow. You are given a set of railroad schedules from which you have to compute the fastest connection among those with the earliest arrival time for going from one location to another. One good thing: Jill is very experienced in switching trains: she can do this instantaneously, i.e., in zero time!!!

Input

The input file contains several scenarios. Each of them consists of three parts.

Part one lists the names of all cities connected by the railroads. It starts with a line containing an integer C $(1 \leq C \leq 100)$ followed by C lines containing city names. These names consist of letters.

Part two describes all the trains running during the day. It starts with a number $T \leq 1000$ followed by T train descriptions. Each train description consists of one line with a number $t_i \leq 100$ and t_i more lines with a time and a city name, meaning that passengers can get on or off the train at that time at that city. The times are given in the 24-hour format `hhmm`.

Part three consists of three lines: Line one contains the earliest possible starting time for the journey, line two the name of the city where she starts, and line three the destination city. The two cities are always different.

The end of the input file is marked by a line containing only a zero (instead of C). Do not process this line.

Output

For each scenario print the line "`Scenario #`*n*" where *n* is the number of the scenario starting at 1.

If a connection exists then print the two lines containing zero padded timestamps and locations as shown in the sample output. Use blanks to achieve the indentation. If no connection exists on the same day (i.e., arrival before midnight), then print a line containing "`No connection`".

After each scenario print a blank line.

Sample Input

```
3
Tuttlingen
Constance
Freiburg
3
2
0949 Tuttlingen
1006 Constance
2
1325 Tuttlingen
1550 Freiburg
2
1205 Constance
1411 Freiburg
0800
Tuttlingen
Freiburg
2
Ulm
Vancouver
1
2
0100 Ulm
2300 Vancouver
0800
Ulm
Vancouver
0
```

Sample Output

```
Scenario #1
Departure 0949 Tuttlingen
Arrival   1411 Freiburg

Scenario #2
No connection
```

ACM International Collegiate Programming Contest 2000/2001
Mid-Central European Regional Contest

Problem H

Smith Numbers

Source: `smith.(c|cc|pas|java)`
Input: `smith.in`

While skimming his phone directory in 1982, Albert Wilansky, a mathematician of Lehigh University, noticed that the telephone number of his brother-in-law H. Smith had the following peculiar property: The sum of the digits of that number was equal to the sum of the digits of the prime factors of that number. Got it? Smith's telephone number was 493-7775. This number can be written as the product of its prime factors in the following way:

$$4937775 = 3 \cdot 5 \cdot 5 \cdot 65837$$

The sum of all digits of the telephone number is $4+9+3+7+7+7+5 = 42$[†], and the sum of the digits of its prime factors is equally $3+5+5+6+5+8+3+7 = 42$. Wilansky was so amazed by his discovery that he named this kind of numbers after his brother-in-law: Smith numbers.

As this observation is also true for every prime number, Wilansky decided later that a (simple and unsophisticated) prime number is not worth being a Smith number, so he excluded them from the definition.

Wilansky published an article about Smith numbers in the *Two Year College Mathematics Journal* and was able to present a whole collection of different Smith numbers: For example, 9985 is a Smith number and so is 6036. However, Wilansky was not able to find a Smith number that was larger than the telephone number of his brother-in-law. It is your task to find Smith numbers that are larger than 4937775!

Input

The input file consists of a sequence of positive integers, one integer per line. Each integer will have at most 8 digits. The input is terminated by a line containing the number 0.

Output

For every number $n > 0$ in the input, you are to compute the smallest Smith number which is larger than n, and print it on a line by itself. You can assume that such a number exists.

Sample Input

```
4937774
0
```

Sample Output

```
4937775
```

[†] What else did you expect???