ACM International Collegiate Programming Contest 2000–2001 Northwestern Europe Regional Contest (NWERC)

TU Darmstadt, Germany

November 19, 2000

Contents

1 Railroads

Background

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

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

Nevertheless, tomorrow in the early morning hours Jill will have to travel from Hamburg to Darmstadt 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 Darmstadt as early as possible. However, she dislikes to get to the station too early, so if there are several schedules with the same arrival time then she will choose the one with the latest departure time.

Problem

Jill asks you to help her with her problem. You are given a set of railroad schedules from which you must compute the train with the earliest arrival time and the fastest connection from one location to another. One good thing: Jill is very experienced in changing trains. She can do this instantaneously, i.e., in zero time!!!

Input

The very first line of the input gives the number of scenarios. Each scenario consists of three parts.

Part one lists the names of all cities connected by the railroads. It starts with a number $1 < C \leq 100$, followed by C lines containing city names. These names consist of letters.

Part two describes all the trains running during a day. It starts with a number $T \leq 1000$ followed by T train descriptions. Each of them 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.

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

Output

For each scenario print a line containing "`Scenario i`", where i 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. 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 and sample output on next page)

Sample Input

```
2
3
Hamburg
Frankfurt
Darmstadt
3
2
0949 Hamburg
1006 Frankfurt
2
1325 Hamburg
1550 Darmstadt
2
1205 Frankfurt
1411 Darmstadt
0800
Hamburg
Darmstadt
2
Paris
Tokyo
1
2
0100 Paris
2300 Tokyo
0800
Paris
Tokyo
```

Sample Output

```
Scenario 1
Departure 0949 Hamburg
Arrival   1411 Darmstadt

Scenario 2
No connection
```

2 Ouroboros Snake

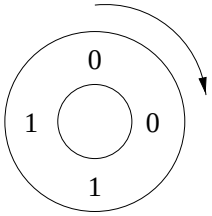
Background

Ouroboros was a mythical snake in Ancient Egypt. It has its tail inside its mouth and continuously devours itself.

Problem

Ouroboros numbers are binary numbers of 2^n bits that have the property of generating the whole set of numbers from 0 to $2^n - 1$ as follows: To produce all of them we place the 2^n bits wrapped in a circle so that the last bit goes before the first one. Then we can denote all 2^n different strings with n bits starting each time with the next bit in the circle.

For example, for $n = 2$ there are only four Ouroboros numbers. These are 0011, 0110, 1100 and 1001. For 0011, the following diagram and table depicts the process of finding all the bitstrings:



k	0011 0011...	$o(n=2,k)$
0	00	0
1	01	1
2	11	3
3	10	2

Your program will compute the function $o(n,k)$, where $n > 0$ and $0 \leq k < 2^n$. This function calculates the k-th number inside the smallest Ouroboros number of size n-bits.

Input

The input starts with a line containing the number of test cases. For each test case you will be given a line with two integers n ($0 < n < 22$) and k ($0 \leq k < 2^n$).

Output

For every test case your output must evaluate the function $o(n,k)$ and print the result on a line by itself.

Sample Input

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

Sample Output

```
0
1
3
2
```

3 Vito's Family

Background

The world-known gangster Vito Deadstone is moving to New York. He has a very big family there, all of them living in Lamafia Avenue. Since he will visit all his relatives very often, he is trying to find a house close to them.

Problem

Vito wants to minimize the total distance to all of them and has blackmailed you to write a program that solves his problem.

Input

The input consists of several test cases. The first line contains the number of test cases.

For each test case you will be given the integer number of relatives r ($0 < r < 500$) and the street numbers (also integers) $s_1, s_2, \ldots, s_i, \ldots, s_r$ where they live ($0 < s_i < 30000$). Note that several relatives could live in the same street number. It could also happen that the optimal street number for Vito coincides with the number of some other relative.

Output

For each test case your program must write the minimal sum of distances from the optimal Vito's house to each one of his relatives. The distance between two street numbers s_i and s_j is $d_{ij} = |s_i - s_j|$.

Sample Input

```
2
2 2 4
3 2 4 6
```

Sample Output

```
2
4
```

4 Smith Numbers

Background

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 type 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 and he excluded them from the definition.

Problem

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 give a Smith number which was larger than the telephone number of his brother-in-law. It is your task to find Smith numbers which are larger than 4937775.

Input

The input consists of several test cases, the number of which you are given in the first line of the input.

Each test case consists of one line containing a single positive integer smaller than 10^9.

Output

For every input value n, you are to compute the smallest Smith number which is larger than n and print each number on a single line. You can assume that such a number exists.

Sample Input

```
1
4937774
```

Sample Output

```
4937775
```

*What else did you expect?

5 Chainsaw Massacre

Background

As every year the Canadian Lumberjack Society has just held its annual woodcutting competition and the national forests between Montreal and Vancouver are devastated. Now for the social part! In order to lay out an adequate dance floor for the evening party the organizing committee is looking for a large rectangular area without trees. Naturally, all lumberjacks are already drunk and nobody wants to take the risk of having any of them operate a chainsaw.

The Problem

The organizing committee has asked you to find the largest yet free rectangle which could serve as the dance floor. The area in which you should search is also rectangular and the dance floor must be entirely located in that area. Its sides should be parallel to the borders of the area. It is allowed that the dance floor is located at the borders of the area and also that trees grow on the borders of the dance floor. What is the maximum size of the dance floor?

The Input

The first line of the input specifies the number of scenarios.

For each scenario, the first line provides the length l and width w of the area in meters ($0 < l, w \leq 10000$, both integers). Each of the following lines describes either a single tree, or a line of trees according to one of the following formats:

- `1 x y`, where the "one" characterizes a single tree, and x and y provide its coordinates in meters with respect to the upper left corner.
- `k x y dx dy`, where $k > 1$ provides the number of trees in a line with coordinates $(x,y), (x+dx, y+dy), \ldots, (x+(k-1)dx, y+(k-1)dy)$.
- `0` denotes the end of the scenario.

The coordinates x, y, dx, and dy are given as integers. It is guaranteed that all the trees are situated in the area, i.e. have coordinates in $[0,l] \times [0,w]$. There will be at most 1000 trees.

The Output

For each scenario print a line containing the maximum size of the dance floor measured in square meters.

Sample Input

```
2
2 3
0
10 10
8 1 1 1 0
8 1 9 1 0
0
```

Sample Output

```
6
80
```

6 Erdös Numbers

Background

The Hungarian Paul Erdös (1913–1996, speak as “Ar-dish”) not only was one of the strangest mathematicians of the 20th century, he was also one of the most famous. 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 the 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 scientific 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.

Problem

Today, nearly everybody wants to know which Erdös number he or she has. Your task is to write a program which computes Erdös numbers for a given set of scientists.

Input

The first line of the input contains the number of scenarios.

The input for each scenario consists of a paper database and a list of names. It begins with the line

```
P N
```

where $P > 0$ and $N > 0$ are natural numbers. Following this line are P lines containing descriptions of papers (this is the paper database). A paper appears on a line by itself and is specified in the following way:

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

Note that umlauts like ‘ö’ are simply written as ‘o’. After the P papers follow N lines with names. Such a name line has the following format:

```
Martin, G.
```

You can assume that not more than 20 authors contribute to a single paper and that a complete paper description is always shorter than 2000 characters.

Output

For every scenario you are to print a line containing a string “`Scenario i`” (where *i* is the number of the scenario) and the author names together with their Erdös number of all authors in the list of names. The authors should appear in the same order as they appear in the list of names. The Erdös number is based on the papers in the paper database of this scenario. Authors which do not have any relation to Erdös via the papers in the database have Erdös number “`infinity`”.

(Sample input and sample output on next page)

Sample Input

```
2
1 1
Hahn, P.M.: My hat is the best
Hahn, P.M.
4 3
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factor matrices
Erdos, P., Reisig, W.: Stuttering in petri nets
Smith, M.N., Chen, X.: First oder derivates in structured programming
Jablonski, T., Hsueh, Z.: Selfstabilizing data structures
Smith, M.N.
Hsueh, Z.
Chen, X.
```

Sample Output

```
Scenario 1
Hahn, P.M. infinity
Scenario 2
Smith, M.N. 1
Hsueh, Z. infinity
Chen, X. 2
```

7 Echo

Background

In the Ancient Caves of Mystery (ACM), scientists have found another old computer. Surprisingly it still worked. The scientists started typing text at the keyboard which was shown immediately on the computer screen. But every now and again some characters were inserted – apparently out of nowhere. After a while they noticed that only characters were inserted which they had typed before. Andrew C. Matthiews, a young and very ambitious theoretician, conjectured that the computer was connected to an interstellar computer network and the inserted characters were in fact an echo generated by some mysterious alien *Echo Race*. The scientists were very excited and immediately started chatting with whoever was sitting (or whatever was convenient to them) at the other side of the network.

They also suspected that such a system must have some finite buffer, so to avoid overloading the system they were careful to wait for a reply when ten characters were entered without remote echo.

Unfortunately, after several hours of work, one of the tired scientists spilled his cup of coffee all over the keyboard. That was too much for the old computer, and it just blew up and could not be revived.

All that the scientists can do now to verify their theory – and such prove the aliens' existence – is looking at their notes of the display contents.

Problem

You are hired to check whether the scientists' notes are consistent with their theory, i.e., if a given string is an "echo string with buffer size ten". Here, an "echo string" e is a merged version of a string s_1 and its identical echo s_2. Note that the order in which the characters of s_1 and s_2 appear in e is not changed. The "buffer size ten" means, that a character in s_1 is not separated by more than nine characters from its echo in s_2.

Due to the sudden end of the experiment, there are three cases:

- The system had echoed all characters when it blew up, so the string really is an echo string with buffer size ten.
- The system blew up before it could echo all characters that were entered, but up to that point its behaviour was consistent with the theory.
- The string cannot be completed by more output to form an echo string of buffer size ten, so the scientists must have entered too many characters at a time or their theory is wrong.

The Input

The first line contains the number n of scenarios.

In each of the following n lines you will find one non-empty string to be examined, and it consists of letters and digits only. No line will contain more than 1000 characters.

The Output

For each scenario, print one of the following, corresponding to the three cases described above:

- `An echo string with buffer size ten`
- `Not an echo string, but still consistent with the theory`
- `Not consistent with the theory`

Sample Input

```
3
ACABCB
ABCAB
aa0123456789b
```

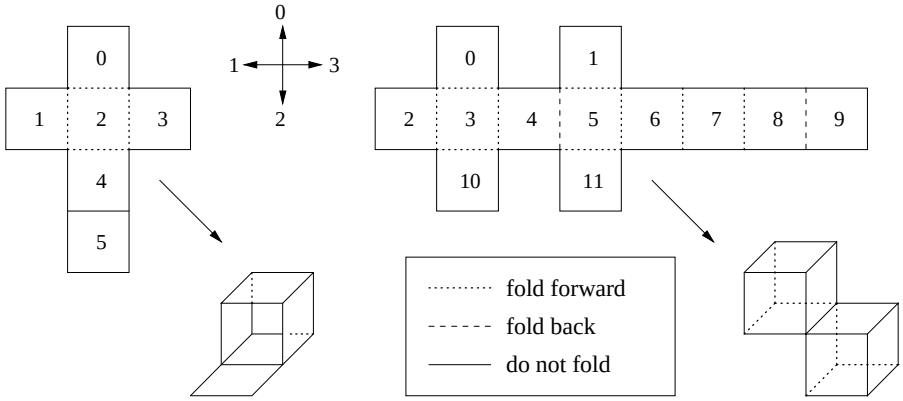
Sample Output

```
An echo string with buffer size ten
Not an echo string, but still consistent with the theory
Not consistent with the theory
```

8 Fold-up Patterns

Background

Fold-up patterns for solids like cubes or octahedrons can be found in many books, but without actually folding them it is hard to tell whether they will really work. We will restrict our attention to a special class of patterns.

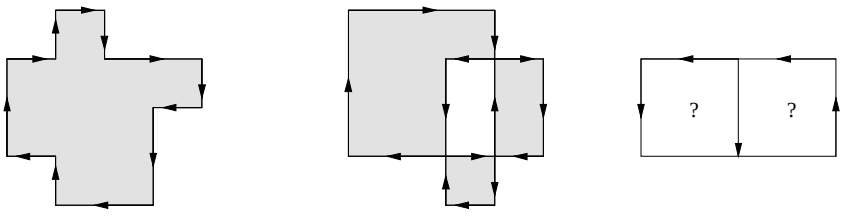


The Problem

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, back, or not at all along that edge, always at right angles. If an edge of 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 so that 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 (the “insides” are shaded).



closed surface | closed surface | not a closed surface

Note that according to our definition the second pattern in the sketch at the top of the page is a closed surface, although it rather looks like two separate cubes attached along an edge.

Two different squares must not occupy exactly the same position in space, though they may (and should 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.

The Input

The first line of the input specifies the number of scenarios.

For each scenario, the first line provides the number $1 \leq n \leq 1000$ of squares in the pattern and the number $0 \leq e \leq 4000$ of edges. Squares are labelled by the integers 0 to $(n-1)$. The following e lines describe one edge each using the following four numbers:

- 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 $p = 0, 1, 2$, or 3 mean above, to the left, below, or to the right, respectively (see sketch).
- The number $f = 0, 1, 2$ tells you to fold along the edge either not at all, or forward, or back, respectively.

It is guaranteed that the folded pattern fits in a cube with a side length of 40. You can also assume that the pattern is connected and can be drawn in the plane without overlapping.

The Output

For each scenario print a line containing 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

```
2
6 5
0 2 2 1
1 2 3 1
2 3 3 1
2 4 2 1
4 5 2 0
12 11
0 3 2 1
1 5 2 1
2 3 3 1
3 4 3 1
4 5 3 2
5 6 3 1
6 7 3 1
7 8 3 1
8 9 3 2
3 10 2 1
5 11 2 1
```

Sample Output

```
Not a closed surface
Closed surface, volume=2
```