ID Hash

University regulations prohibit public posting of grades if individual students are easily identifiable. For example, ID numbers should not be shown explicitly. One may, however, use a portion of the ID if reasonable effort is made to disguise the posted number.

Consider the following hashing scheme based upon a positive integer m.

> For each ID in the class, take the number represented by the last 4 digits of the ID and reduce it modulo m.

Write a program that, given the ID's for the class, determines the minimum m for which the hashing is free of collisions (duplicate values).

The input to your program will be a sequence of classes, given by student ID's. The first line for each class will indicate the class size. Following that, the ID's will appear in the usual format ($$$-$$-$$$$), one to a line. (Class size will be at most 25.)

The output should be the minimum hash modulus m for the class if one exists. If no such m exists the output then the output should read "`Collisions are unavoidable!`"

Sample Input

```
3
109-45-4361
083-45-6680
924-98-0135
4
874-35-7447
331-88-9285
078-12-7447
991-67-5712
```

Sample Output

```
4
Collisions are unavoidable!
```

Observe for the first class that, modulo 2, the numbers 4361, 6680, 135 are respectively congruent to 1,0,1; viewing them modulo 3, they reduce to 2,2,0; however, modulo 4 they yield three distinct values 1,0,3. For the second class, there is no collision-free modulus since two of the numbers formed by taking the last 4 digits are identical.

SubDuoPalindromes

A *subduopalindrome (SDP)* in a string S is a pair of substrings $\{S_1, S_2\}$ (not necessarily disjoint) of S such that S_2 is the reverse of S_1. The *size* of such an SDP is the total number of string positions involved in either S_1 or S_2.
NOTE: The size is determined by the *union* of the string positions; S_1 and S_2 may overlap, and positions common to S_1 and S_2 are only counted once. See the sample input/output for examples.

Write a program that will accept a string S as input and determine the maximum size SDP's of S.

Input will be a sequence of strings, each of length at most 25, one per line.

For each input string S, one line of output should indicate the SDP of maximum size. For each such pair, S_1 should be printed, followed by a space, followed by S_2 *even if* S_1 *and* S_2 *overlap within* S. (In an SDP $\{S_1, S_2\}$, if $S_1 \neq S_2$, then S_1 is taken to be the leftmost of the two.) Successive maximum size SDP's of a string, if there are more than one, should be separated by a space and ordered as follows: $\{S_1, S_2\}$ should precede $\{S_1', S_2'\}$ if either the first character of S_1 precedes the first character of S_1', or S_1 and S_1' have the same first character and the last character of S_2 precedes the last character of S_2'.

Sample Input

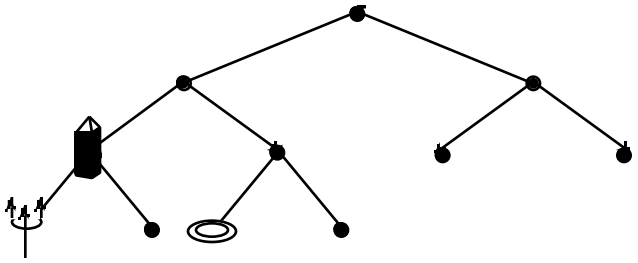
```
abc
defgedeffgefed
hijjjiklmnopkonmmno
rsstutssr
vwxxwzzzzywv
```

Sample Output

```
a a b b c c
def fed def fed
mno onm onmmno onmmno
rsstutssr rsstutssr
vw wv wxxw wxxw zzzz zzzz
```

Spirited Binary Trees

A *spirited binary tree (SBT)* may include, in addition to its *ordinary* nodes, three special kinds of nodes: *milestones*, *devils*, and *angels*.



The angels and devils meddle in your preorder traversal. At ordinary nodes, normal preorder sequencing is followed (you visit the node first, then its left subtree, and then its right subtree). But, supernatural events take place when you visit a special node:

- When visited, a milestone acquires a *charge*.
- When visited, a devil implants in your subconscious a *summons* to return and then teleports you back to the latest *charged* milestone, at which point that milestone loses its charge. The preorder traversal continues as if the subtree rooted at the milestone has just been traversed (in particular, this is not considered a visit to the milestone). If no milestone is currently charged, a devil has no effect
- When visited, an angel teleports you to the devil with the most recently implanted summons, at which point that summons is released from your subconscious. The preorder traversal continues as if the subtree rooted at the devil has just been traversed (in particular, this is not considered a visit to the devil). If no summons is currently implanted, an angel has no effect.

Write a program to traverse an SBT in preorder, printing the nodes in the order visited.

The input has the following format. Nodes are numbered starting 1. Each node is represented on a line by four critical elements separated by spaces: first comes the ID number of the node; next a label describing the node as `ordinary`, `milestone`, `devil`, or `angel`; then the ID numbers of the left child and right child. The root is the node on the first line. A leaf has children 0 and 0.

The output should list, in order and one to a line, the number of the visited node, followed by a space, then followed by the node's label.

Sample Input (for pictured tree)

```
1 ordinary 2 3
2 ordinary 4 5
3 ordinary 6 7
4 milestone 8 9
5 ordinary 10 11
6 ordinary 0 0
7 ordinary 0 0
8 devil 0 0
9 ordinary 0 0
10 angel 0 0
11 ordinary 0 0
```

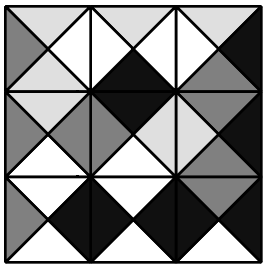
Sample Output

```
1 ordinary
2 ordinary
4 milestone
8 devil
5 ordinary
10 angel
9 ordinary
5 ordinary
10 angel
11 ordinary
3 ordinary
6 ordinary
7 ordinary
```

TetraTiles

TetraTiles are formed by joining four congruent colored isosceles right triangles to form a square. Four colors are used for the triangles and the individual TetraTiles might use these in any combination. Some sample types:

In using the TetraTiles for a tiled floor pattern, it is desirable to match colors where TetraTiles have a common edge. Furthermore, nine TetraTiles types are selected to form a fundamental 3×3 SuperTetraTile. The SuperTetraTile also must have single-colored sides, with each of the four colors comprising one of the sides. (Copies of this SuperTetraTile can be used to tile the floor in a periodic pattern.) With appropriate rotations, it is possible to form a SuperTetraTile with the above nine TetraTiles.



Write a program that will determine whether a given set of nine TetraTiles can be used to form a SuperTetraTile

Each line of the input specifies nine TetraTiles to test for supertetratilability. An individual TetraTile is given by indicating the colors of its four triangles, read in clockwise order, and are separated by a single space. e.g. `R B B G` (note: by rotation, this is identical to `B G R B`). A single space separates TetraTiles specifications from one another. The possible colors are: `R`, `W`, `B`, and `G`.

If the nine TetraTiles are supertetratilable, one possible SuperTetraTile should be output. The SuperTetraTile should always be oriented with `G` on the top edge. The SuperTetraTile is then specified by indicating, in succession, the colors in the TetraTiles on the top row, left to right, then the middle row and, finally, the bottom row similarly. An individual TetraTile is indicated by giving the colors in clockwise order, always starting with the top-edge color as it is oriented in the the SuperTetraTile. If the nine TetraTiles *cannot* be configured to form a SuperTetraTile, the output should be "Unsupertetratilable!"

Sample Input (the first is for tiles pictured above)

```
R W G B R B G W W B W R W G R G W B W B G R W R W B R B B R G R W G W B
W W W W R R R R B B B B G G G G R R B W B B W R W W R B B R G W B W G R
```

Sample Output

```
G W G R G W B W G B R W G R W R B G W R R B R G W B W R W B B W R B W B
Unsupertetratilable!
```

Lucky Medians

To find the median of an array, $A[1..n]$, of n distinct numbers, it is not necessary to sort A. There are fast methods that guarantee only $O(n)$ comparisons, though these tend to be complicated. Sometimes, however, you can get lucky. For example, if the array is $A[1..5]$ and you have learned that $A[2] < A[3]$, $A[3] < A[1]$, $A[4] < A[2]$, and $A[5] < A[2]$, then you already know that $A[2]$ is the median element since half the elements are demonstrably less than $A[2]$ and half are greater than $A[2]$.

Write a program that finds whether a specified collection of results of comparisons suffices to determine the median of the array and, if so, indicate the index of the median element. The length of each array will be odd and at most 25.

Input will be results of comparisons on some sets. The data for each set begins with a line giving the length of the array; then a line indicating the number of comparisons to be reported; the comparisons follow, one to a line, where "i j" indicates $A[i] < A[j]$.

Output for each array is "m" if it can be determined that $A[m]$ is the median element or "`unlucky`" if the median cannot be deduced from the given data.

Sample Input

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

Sample Output

```
2
unlucky
```

Pebble Pushers

A game is played with a pebble on a directed acyclic graph, $G = (V, E)$.

The pebble is initially situated on some $v_0 \in V$. Two players A and B then move in turn. A player may push a pebble along any arc (i.e., if $(v, w) \in E$, a pebble on v may be pushed to w). The last player able to move wins the game.

Write a program which you will you use to determine whether you want to move first or second, and, if you move first, what your first move would be (there may be more than one correct answer in that case).

Input will be a sequence of initial game setups. The data for each game is organized as follows: the (≤ 26) vertices of the graph will be always be an initial seqment of the alphabet $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \ldots$; the first line for the game is a pair of numbers $|V|$, $|E|$ (number of vertices, number of edges) separated by a single space; the second line indicates the initial position of the pebble; the next $|E|$ lines give the the edges, each given as a pair of vertices separated by a space (i.e., "`r s`" indicates that $(\mathtt{r}, \mathtt{s}) \in E$).

Output should be any vertex to which you would move if it is best to move first or else the phrase "`I want to go second.`"

Sample Input

```
4 4
b
a c
b d
d a
b c
3 2
a
a c
c b
```

Sample Output

```
d
I want to go second.
```

Periodic Binaries

As you know, the decimal expansion of a rational number is eventually periodic. This, of course, holds for binary expansions as well.

Write a program that will accept a pair of positive integers (≤ 1000) and determine the length of the period in the binary expansion of m/n.

Input will be a sequence of pairs of numbers, one pair per line with a space between the numbers in each pair.

There should be one line of output for each input, containing just the length of the period in the binary expansion of the rational number.

Sample Input

```
3 1
15 9
13 8
9 28
```

Sample Output

```
1
2
1
3
```

The above output corresponds to the binary representations:

$$\left(\frac{3}{1}\right)_2 = 11.00000000000000\cdots = 11.\overline{0}$$

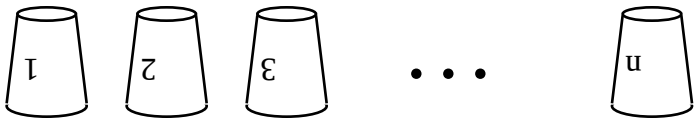
$$\left(\frac{15}{9}\right)_2 = 1.10101010101010\cdots = 1.\overline{10}$$

$$\left(\frac{13}{8}\right)_2 = 1.101000000000000\cdots = 1.101\overline{0}$$

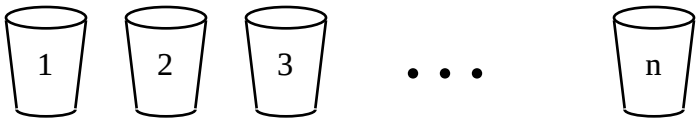
$$\left(\frac{9}{28}\right)_2 = 0.01010010010010\cdots = 0.01\overline{010}$$

Flippant Cups

A puzzle involves a line of paper cups, all initially turned upside down



and the objective is to flip them all right side up.



However, the legal moves are restricted to a collection of triples, where the triple i, j, k represents a simultaneous flip of cup i, cup j, and cup k.

Write a program that will accept a set of legal moves and determine whether the puzzle can be solved.

Input will be a sequence of puzzle specifications. A specification begins with a line containing two numbers, the number of cups (up to 20) and the number of legal moves, separated by a space; the legal moves then follows as labeled triples, one to a line, namely, a single-character label then a space and then three numbers separated by spaces.

Output for each puzzle is a set of labels for moves that solve the puzzle or else "`Unsolvable`", with one answer per line.

Sample Input

```
4 2
a 1 2 3
b 1 3 4
5 4
c 1 3 5
d 1 2 3
e 2 4 5
f 1 3 4
```

Sample Output

```
Unsolvable
c d f
```

Notice that the second puzzle is solved by performing moves $(1, 3, 5), (1, 2, 3), (1, 3, 4)$.