Problem A: Crypto-Math

A common puzzle is to present a math problem where each digit is replaced by a letter. So, for example the sum:

$$\begin{array}{r} 112 \\ \underline{+\,234} \\ 346 \end{array}$$

could be represented as:

$$\begin{array}{r} \text{AAB} \\ \underline{+\text{BCD}} \\ \text{CDE} \end{array}$$

where A=1, B=2, C=3, D=4, and E=6. Notice that the same digit always replaces all instances of the same letter. It will also be the case that each distinct letter will be replaced by a different digit.

Your task is to take a problem written as letters, and display the equivalent version using numbers.

Input:
There will be several input instances. Each input instance will begin with a number, N, indicating the number of different letters used in the math problem. A value of N=0 will indicate the end of input. On the next line will be the problem, in the form (for example): AAB + BCD = CDE. All letters will be capital, and only the first N letters (starting from A) will be used. Each operand (and the sum) will be at most 5 letters long.

Output:
For each input instance, output the equivalent mathematical statement, after replacing letters with numbers. If there is more than one possible answer for a given input, display the one with the lowest digit for "A", then the lowest digit for "B", and so on. If there is no possible way to assign digits legally to solve the problem, then output "No solution possible"

Sample Input:

```
5
AAB + BCD = CDE
9
EFGH + ABCD = BIEF
3
AAA + B = BDDD
4
AAA + B = CDDD
0
```

Sample Output:

```
112 + 234 = 346
2769 + 0358 = 3127
999 + 1 = 1000
No solution possible.
```

Problem B: Three-card poker

Consider the following simplified card game. The deck only has cards from 1 (low) to 9(high) (i.e. no tens, jacks, queens, or kings, and aces are treated as 1's), and all four suits are represented (clubs, diamonds, hearts, and spades).

In this game, each player gets three cards, and makes the best hand possible. The ranking of the hands is slightly different from regular poker:

- The highest hand is a *straight flush*, where the player has three consecutive cards of the same suit. The 4,5, and 6 of clubs is a straight flush.
- The next highest hand is a *three of a kind*, where the player has three of the same number. The 9 of clubs, 9 of spades, and 9 of hearts would make a three of a kind
- The next highest hand is a *flush*, where the player has three of the same suit (but not consecutive, because that would be a straight flush). The 1,5, and 8 of diamonds would be a flush.
- The next highest hand is a *straight*, where the player has three consecutive cards (but not all in the same suit, because that would be a straight flush). The 2 of diamonds, 3 of diamonds, and 4 of hearts would be a straight.
- The lowest hand is a *nothing*, which catches any hand that doesn't make one of the above qualifications. (Notice that a pair is a nothing hand by these rules)

All hands in one ranking are better than all hands in a lower ranking, regardless of the actual cards that make the hand. In other words, a three of a kind made out of 1's beats a straight (but not a straight flush) made out of 9,8, and 7.

If two hands are in the same ranking, the higher hand is the one that has the highest total of numbers on the cards.

Your task is to read in two hands (where the cards in a hand can be arranged in any order), and to determine which one is higher, or if they are tied.

Input:
There will be several input instances. Each input instance will list 6 cards. Each card will be of the form NS, where N is the number of the card (a digit between 1 and 9), and S is the suit (one capital letter denoting the first letter of the suit). The first three cards will comprise the first hand, and the next three cards will be the second hand. A first card with a numeric value of 0 indicates the end of input

Output:
For each input instance i, output the line:

```
Hand i: Player 1/2 wins.
```

Player 1 is the first hand, player 2 is the second hand.

If there is a tie, output the line:

```
Hand i: There is a tie.
```

Sample input:

```
9C 9D 9H 9S 1S 2D
9C 9D 9H 1C 1D 1H
2D 3H 4H 2S 4S 3S
2S 3H 7S 5S 5S 2C
0D 0D 0D 0D 0D 0D
```

Sample Output:

```
Hand 1: Player 1 wins.
Hand 2: Player 1 wins.
Hand 3: Player 2 wins.
Hand 4: There is a tie.
```

Problem C: **Gold Rush**

Being the mayor of the sleepy town of Aardvark Creek, Montana was an easy job. Easy, that is, until someone found gold in the outskirts of town. Now, mobs of treasure-seekers have overrun the town, each staking a claim to some area of the wilderness. What's worse is that these gold diggers have often claimed overlapping areas, and have come to you to resolve who owns what.

Thanks to an old bylaw that has been on the books for centuries, all claims have been recorded in triangular areas, defined by three vertices. Your job is to analyze the vertices that define a pair of claims, and to determine how many vertices from each claim fall within the area of the other claim. Generally, the claim with the most "intruding points" will be found at fault.

Your job is to analyze several pairs of claims, and to categorize them based on the above criteria.

Input:
The first line of the input will be an integer n, denoting the number of input instances. There will then follow n lines of input, each line consisting of 12 integers. Each line will be of the form
x_{11} y_{11} x_{12} y_{12} … x_{23} y_{23}
where x_{ij} denotes the x coordinate of vertex j in claim i, and y_{ij} represents the corresponding y coordinate. All coordinates will be positive integers. Each claim will have positive area (i.e. the three vertices of a claim will not be collinear). The two claims will have no vertices in common, and no vertex will lie along a line defined by two vertices of another claim.

Output:
For each input instance i (starting at 1), output one of the following lines:

```
Case i: Claim a has x vertices inside Claim b.
```

Where a is the claim (1 or 2) with the most vertices inside the other, b is the number of the other claim, and x is the number of vertices inside the other.

If both claims have the same number of intruding points, output:

```
Case i: Both claims have equal numbers of intruding points: x
```

Where x is the number of intruding points.

Sample Input:

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

Sample Output:

```
Case 1: Both claims have equal numbers of intruding points: 0
Case 2: Claim 1 has 3 vertices inside Claim 2.
Case 3: Claim 2 has 1 vertices inside Claim 1.
```

Problem D: Ultimate Loser

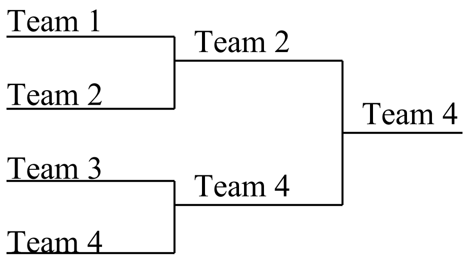
In single-elimination tournaments (for example, the NCAA Basketball tournament), teams that win play each other until there is only one team left, and that team is named the champion.

For the pessimists among us, it's also fun to figure out who the worst team is. Since half of the entrants will lose their first game, choosing between them is difficult. Consider the following algorithm:

- Find the team that lost in the championship
- Find the team that lost to that team in the semifinals
- Find the team that lost to that team in the previous round
- Continue this process until you get back to the first round. That team is the "ultimate loser", because they are the end of a chain of teams that lost as soon as possible.

(the ultimate loser in the 2007 NCAA basketball tournament was New Mexico State. So feel free to bring that up if you know someone who goes there)

Here is a small example:



In this case, team 4 is the champion, and team 1 is the ultimate loser, because 2 lost to 4, and 1 lost to 2.

Input:
There will be several input instances. Each instance will start with a number, n, indicating the number of teams. A value of n=0 will indicate end of input, otherwise n will be a power of 2, between 2 and 64 (inclusive). The value of n will indicate a bracket similar to the 4-team one above, with teams numbered from 1 to n, listed in the bracket sequentially. Teams 1 and 2 will always play in the first round, and will play the winner of teams 3 and 4 in the next round, and so on.

The next line will be n-1 integers, indicating the winners of each game, reading the bracket from top to bottom. All first round winners will be listed first (there will be n/2 of these), then the n/4 second round winners, and so on until the champion is listed. So, for example the first number listed will either be a 1 or 2, since the first game is always between teams 1 and 2.

Output:
For each input instance i, output the line:

```
Case i: The ultimate loser is team x.
```

where "x" is the number of the team that is the ultimate loser.

Sample Input:

```
4
2 4 4
8
2 4 5 7 2 7 2
0
```

Sample Output:

```
Case 1: The ultimate loser is team 1.
Case 2: The ultimate loser is team 6.
```

Problem E: Superdelegates

As the Democratic Party's nomination process continues, we've been hearing about how their nomination process has two facets:

- "committed delegates", who are selected by the primary and caucus processes in each state
- "superdelegates", who are free to choose any candidate they want.

Suppose, for the sake of simplicity, that there are only two candidates for the nomination. When the convention comes around, all of the committed delegates will be allocated to one candidate or the other. Additionally, some of the superdelegates have pledged their support to one candidate or another. The candidates then try to sway enough of the superdelegates that have not yet pledged their support to their side, in an effort to secure the majority of the delegates (and thus win the nomination).

You're working in the office of Candidate A, who's running against Candidate B. Given the total amounts of committed delegates, and the current situation of the pledged superdelegates, you need to figure out how many of the unpledged superdelegates you need to win over to your side to have a majority of delegates overall.

Input:
There will be several input instances. Each instance will consist of five non-negative integers on a line:
ca cb pa pb u

ca = committed delegates for candidate a
cb = committed delegates for candidate b
pa = pledged superdelegates for candidate a
pb = pledged superdelegates for candidate b
u = unpledged superdelegates

Each value will be an integer < 10,000. A value of *ca* < 0 indicates end of input

Output:
For each input instance, output the line:

```
Candidate A needs at least x of the unpledged superdelegates.
```

..where x is the smallest number of unpledged superdelegates needed to achieve a majority of the total delegates. The value of x should never be negative.

If Candidate A can't gain a majority even with all of the unpledged superdelegates, output the line:

```
Candidate A has lost the nomination!
```

Sample Input:
```
10 10 10 10 10
1 2 3 4 5
10 100 10 100 10
-1 –1 –1 –1 -1
```

Sample Output:
```
Candidate A needs at least 6 of the unpledged superdelegates.
Candidate A needs at least 4 of the unpledged superdelegates.
Candidate A has lost the nomination!
```

Problem F: Wikipedia Dominance

In Wikipedia, articles on topics often contain links to other topics. Suppose we want to find out what the "most important" topic on Wikipedia. One way to define that is through the concept of "dominance", using the following rule:

If topic A contains a reference to topic B, but topic B doesn't contain a reference to topic A, topic B dominates topic A (in other words, B is more important because A needs to link to B to be complete, but B doesn't need a link to A).

In a real-life example, "Asia" dominates "apples", because the Wikipedia entry for apples contains a link to Asia, but the entry for Asia doesn't contain a link to apples. If the Asia article had a link to apples, then there would be no dominance between the two articles.

One potential problem with this method of finding more important articles is that it is possible for there to be a circular chain of articles dominating each other (i.e. A dominates B, B dominates C, and C dominates A). We'd like to detect this situation if it arises. Your task is to look at a series of articles and find out if this circular dominance situation exists.

Input:
There will be several input instances. Each instance will begin with an integer, n, denoting the number of articles to consider, $n \leq 20$. A value of n=0 will mark the end of input.

If $n \geq 1$, there will then follow n lines. Each line will begin with a string, which will be the name of the article. Then there will be an integer m, which will be the number of references in the article ($0 \leq m \leq n-1$, since an article can't reference itself). Then there will be m strings, which will be the names of the articles referred to from the current page.

Each string will have no whitespace in it, and all strings in references will be listed as a page on one of the n lines (though it's possible for a page to not be referenced at all).

Output:
For each input instance i, output either:

```
Case i: There is a dominating circle.
```

or

```
Case i: There is no dominating circle.
```

..whichever is appropriate

Sample Input (These examples are not accurate on Wikipedia):

```
5
Apple 2 Banana Cow
Banana 1 Cow
Cow 1 Dallas
Dallas 2 Apple Banana
Elephant 3 Apple Banana Cow
7
Algeria 2 Bali Canada
Bali 2 Denmark Ecuador
Canada 2 France Germany
Denmark 0
Ecuador 0
France 0
Germany 0
3
Alien 1 FBI
FBI 1 Moon
Moon 1 FBI
0
```

Sample Output:

```
Case 1: There is a dominating circle.
Case 2: There is no dominating circle.
Case 3: There is no dominating circle.
```