2017 Denison Spring Programming Contest
Granville, Ohio
18 February, 2017

Rules:

1. There are **six** problems to be completed in **four hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. No whitespace should appear in the output except between printed fields.
4. All whitespace, either in input or output, will consist of exactly one blank character.
5. The allowed programming languages are C, C++, Python 2, Python 3, and Java.
6. All programs will be re-compiled prior to testing with the judges' data.
7. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
8. The input to all problems will consist of multiple test cases.
9. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
10. All communication with the judges will be handled by the PC^2 environment.
11. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Bucket Twist

You're at a lake and have to measure out exactly t units of water using three buckets given to you. The buckets have integer capacities of a, b, and c. (Note: These may not be distinct.) You can perform the following operations:

1. Fill up a bucket from the lake.
2. Pour as much as you can from one bucket into another (i.e., until the first is empty or the second is full).
3. Pour the entire contents of one bucket back into the lake.

You have to figure out if it is possible to get exactly t units of water into any one of the buckets and the minimum number of the above operations this will take. For example, if your goal is to obtain 7 units of water and you have buckets of capacity of 2, 6, and 11, then you can do this by

- filling the 11-unit bucket;
- pouring the 11-unit bucket into the 6-unit one (leaving 5 in the former);
- filling the 2-unit bucket; and
- pouring the 2-unit bucket into the 11-unit bucket, resulting in 7 units in the latter.

Therefore, it is possible to get 7 units in one of the buckets with 4 operations. (By the way, the solution isn't unique—you could also empty out the 11-unit bucket twice into the 2-unit one, again using 4 operations.)

With the same bucket sizes, it is possible to get 8 units by filling the two smaller ones and emptying them into the larger (4 operations). Or to get 4 units by filling the 6-unit bucket and pouring out 2 units into the smallest one (2 operations).

Input

Each input will consist of a single line with four integers t a b c, where t is the target amount and a, b, c are the bucket capacities. All inputs are integers between 1 and 100, inclusive ($1 \leq t, a, b, c \leq 100$). A line with 0 0 0 0 will terminate the input cases.

Output

For each input, you should output the minimum number of operations that it takes to obtain the target quantity in one of the buckets, or the word `impossible` if it is not possible, using the format given in the Sample Output.

Sample Input

```
7 2 6 11
7 11 6 2
8 2 6 11
4 2 6 11
12 2 6 11
7 2 4 8
0 0 0 0
```

Sample Output

```
Case 1: 4
Case 2: 4
Case 3: 4
Case 4: 2
Case 5: impossible
Case 6: impossible
```

Problem B: Elevators

You are riding the lone elevator in the exclusive Harrumph Towers, which houses large corporations, law firms, advertising firms, and consulting practices. Each company occupies an entire floor and as the elevator stops and the doors open, you see the name of the firm on the wall.

Unfortunately, the elevator in Harrumph Towers is not well-maintained and the lights indicating the floors are all out, so you never really know which floor you are on. You can tell, of course, if the elevator is traveling up or down. But the elevator itself is a bit erratic and you can not tell the distance the elevator travels. For a few hours' amusement, you simply ride the elevator as others press buttons and get on and off. As the doors open, you observe the names of the floors. Successive stops do tell you the relative position of the offices of two companies: company A is above, or below, company B. But that is all. After a few stops, you can sometimes deduce the relative floors of some of the companies.

For example, suppose you start in the floor with company A, go up to B, up to C, down to D, up to B (again), down to E. Then you know, among other things, that A, D, and E are below B and C. However, you can't really tell the relative positions of A, D, and E.

Input

Each input consists of three lines. The first line has a single integer n ($2 \leq n \leq 100$) indicating the number of floors visited. (A value of $n = 0$ indicates the end of the input.) The second line is of the form $s_1\ d_1\ s_2\ d_2\ \ldots\ s_n$, where the s_k are strings (with no spaces in them), indicating the company names observed when the doors open, and the d_k are the characters `U` or `D`, indicating the direction next travelled. After this there is a line of the form $m\ t_1\ w_1\ t_2\ w_2\ \ldots\ t_m\ w_m$ ($1 \leq m \leq 100$). Each pair $t_k\ w_k$ is a query asking the relative position of company t_k to w_k. You may assume each t_k and w_k have been visited, that is, each is one of the s_k. You may also assume that there are no physically impossible inputs (e.g., ones in which a floor is above itself).

Output

The answers to the queries for each input set should appear on one line, separated by a space, using the format given in the Sample Output. The answers to each query should be a single character, either `<`, `>`, or `?`, indicating, respectively, that t_k is on a lower floor than w_k, t_k is on a higher floor than w_k, or that it is impossible to determine their relative position.

Sample Input

```
6
Hardware U Betty's U InternationalCorp D Denny's U Betty's D Eddie'sGarage
3 Hardware Denny's Eddie'sGarage Betty's InternationalCorp Hardware
6
Stark D Frey D Lannister D Baratheon D Martell U Targaryen
4 Stark Baratheon Martell Stark Baratheon Targaryen Targaryen Martell
5
WeasleysWizardWheezes U Honeydukes D DailyProphet D Ollivanders D Gringotts
3 DailyProphet Gringotts Honeydukes Ollivanders WeasleysWizardWheezes Gringotts
6
A U B U C D D U B D E
6 A B D C B E D E A D A E
0
```

Sample Output

```
Case 1: ? < >
Case 2: > < ? >
Case 3: > > ?
Case 4: < < > ? ? ?
```

Problem C: iPod Bingo

All the cool college kids are playing a new party game called iPod Bingo. The game starts by randomly selecting someone's mp3 player. The mp3 often has many dozens of songs and also several playlists that group the songs into collections. Someone at the party picks a starting song and an ending song from the mp3 player. The goal is to play the start song, the end song and as few songs in between as possible. Here are the rules of the game.

After playing the start song, the DJ can play any song next that shares a playlist with the start song. After picking song 2, the DJ can play any song that shares a playlist with song 2. The game continues, song by song, until the DJ is able to play the finishing song. Being a computer scientist in her spare time, the DJ wants to automate the process of figuring out the fewest songs that have to be played.

Consider an example with $n = 5$ songs and $m = 4$ playlists as shown in this figure. If the start song is 1 and the end song is 4, then the DJ can play Song 1, then Song 2 (they share Playlist 1), then Song 3 (Song 2 and Song 3 share Playlist 2), then Song 4 (which shares Playlist 3 with Song 3). That is a total of 4 songs. There is no shorter way to get from Song 1 to Song 4.

```
Song 1 --(Playlist1)--> Song 2 --(Playlist2)--> Song 3 --(Playlist3)--> Song 4
```

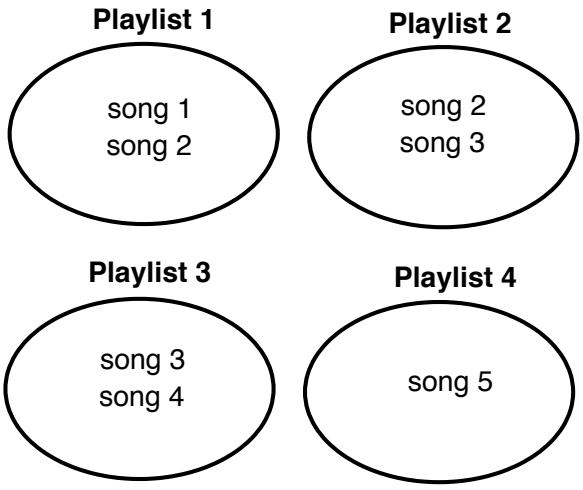

Figure 1: Example MP3 Player

Input

The input for each problem instance is given by a pair of numbers, n and m, denoting n songs (numbered between 1 and n) and m playlists on the mp3 player. These values will be bounded by $1 \leq n \leq 300$ and $1 \leq m \leq 50$. The next line of input will contain two numbers s and e, both $1 \leq s, e \leq n$, indicating the number of the start song and the end song. The playlists are given in m lines of input. Each playlist description starts with the number of songs on the playlist (between 0 and 30) followed by that number of songs separated by spaces. The input will end with the line 0 0.

Output

For each line of input (each problem) you should output the answer or the words "not possible". The answer indicates the fewest total number of songs needed to jump from the start to the end song (this

count includes both the start and end songs). If there is no way to transition between the start and end songs, print "not possible" as the output. Follow the Sample Output for formatting.

Sample Input

```
5 4
1 4
2 1 2
2 2 3
2 3 4
1 5
5 4
1 2
2 1 2
2 2 3
2 3 4
1 5
5 4
1 5
2 1 2
2 2 3
2 3 4
1 5
7 5
1 7
2 1 2
3 2 3 4
2 3 5
3 7 6 5
2 1 6
0 0
```

Sample Output

```
Case 1: 4
Case 2: 2
Case 3: not possible
Case 4: 3
```

Problem D: Missiles

You're designing a missile simulation game in which missiles travel on a two-dimensional grid. The missiles can be considered points and start out at integer coordinates on the grid. A pair of missiles annihilate each other if they meet at the same point at the same time. (You may assume that three or more missiles never meet at the same point.) Your task is to determine how many (and which) of the missiles survive after all such collisions have occurred.

Input

Each input consists of multiple lines. The first line will have a single integer n ($1 \leq n \leq 100$) indicating the number of missiles. A value of 0 indicates the end of the input. The subsequent n lines will have four integers x y v_x v_y indicating the starting position (x, y) of the missile on the grid ($-1000000 \leq x, y \leq 1000000$) and the velocity of the missile in the x and y directions v_x and v_y, respectively ($-100 \leq v_x, v_y \leq 100$). For example, values of $v_x = 1$ and $v_y = -2$ indicate that the missile moves, in a straight line, one unit to the right and two units down on the grid every second. You may assume that each missile originates at a distinct point.

Output

For each input, you should output the number of missiles that never collide with any others, followed by the identity of the surviving missiles in sorted order (the first listed in the input is 1, the next is 2, etc.), using the format in the Sample Output. Keep in mind that the missiles may collide at non-integral points and even off the grid described above.

Sample Input

```
2
0 0 1 0
10 0 -1 0
3
0 0 1 0
10 0 -1 0
1 0 0 -1
5
0 0 1 1
3 0 -1 1
0 -100 0 5
0 100 0 -7
100 100 1 1
4
1 0 1 0
-1 0 -1 0
0 1 0 1
0 -1 0 -1
0
```

Sample Output

```
Case 1: 0
```

```
Case 2: 1 3
Case 3: 1 5
Case 4: 4 1 2 3 4
```

Problem E: Recipe Ratios

You've gotten a recipe for alfredo sauce that has several different ingredients and you want to know how much sauce it will make. Instead of solving this one problem, you set out to generalize your solution so that it can answer the question for any recipe.

For example, if the alfredo recipe calls for 2 cups of milk, 3/4 cup of cream cheese, and 3/2 cups of Parmesan cheese, then the total amount produced by the recipe is $2 + 3/4 + 3/2 = 17/4$ cups.

Input

Each input will consist of two lines. The first line will have the number of ingredients in the recipe n ($1 \leq n \leq 100$). (An input of $n = 0$ indicates the end of input.) The following line will have n pairs of positive integers indicating the fractions of cups of each of the n ingredient in the recipe. Each integer will be between 1 and 1000, inclusive. The products of the denominators will not exceed two billion.

Output

For each input, you should output the total amount produced by the recipe given as a fraction, as shown in the Sample Output. The fraction should be reduced to lowest terms (e.g., 2/3 instead of 4/6). The numerator and the denominator of any unreduced answer will both be less than two billion.

Sample Input

```
3
2 1 3 4 3 2
3
2 1 2 4 3 2
4
4 1 3 1 2 1 1 1
2
1 2 1 3
4
1 100 1 100 1 100 1 100
0
```

Sample Output

```
Case 1: 17/4
Case 2: 4/1
Case 3: 10/1
Case 4: 5/6
Case 5: 1/25
```

Problem F: Sweet Poppers

Sweet Poppers are candy balls that come packaged in a tube of 10 poppers. The candies are removed from the top of the tube in the reverse order they were packaged. There are five different colored candies: red, blue, green, purple, and yellow (denoted r, b, g, p, and y). The manufacturers have rules about packaging Sweet Poppers into a tube:

1. There can be at most five green candies in a package.
2. It is not permissible to have more than two purple candies in a row.
3. Any blue candies must be packaged lower in the tube than all the red candies (all red must come out before any blue).

Your job is to examine a package of Sweet Poppers and determine if they are packaged correctly according to these rules.

Input

The input starts with the number n indicating n different Sweet Popper packages (n different problems). The input is then given on n subsequent lines, one line of input per problem/package. Each problem is given by a string of 10 characters consisting of lower case letters r, b, g, p, and y. The front of the string corresponds to the top of the tube package.

Output

For each input you should output either `yes` or `no` denoting if the packaging for that problem is correct or not, as shown in the Sample Output.

Sample Input

```
4
grgrggrggg
rrbgrprbbb
rprggrbpgb
ggyypprbbb
```

Sample Output

```
Case 1: no
Case 2: no
Case 3: yes
Case 4: yes
```