The English variant of the Hawai‘ian language has the 5 vowels \{a, e, i, o, u\} and the 7 consonants \{h, k, l, m, n, p, w\}. There’s also that weird backwards apostrophe which signifies a break when you sound out a word.

And sounding out those words can be nasty sometimes, mostly because of the vowels. Here are some incontrovertible rules for Hawai‘ian words:

- Every word ends in a vowel.
- Words never have two or more consonants in a row.
- A “w” actually makes a “v” sound when it comes before any of “a”, “e” or “i”.
- A consonant begins a sound, but never appears in the middle or the end of a sound.
- There is always a break in sound between two of the same vowel in a row.
- The vowel combinations “ae”, “ao”, “au”, “ei”, “oi”, and “ou” are always pronounced as part of the same sound, regardless of what letter came before it.
- The vowel combination “ai” is pronounced as part of the same sound if and only if it begins the word, or it is immediately preceded by a “k”, “m” or “w”.
- Last, but not least, the letter combination “kiu” is pronounced “tsiu”, all one sound.

The pronunciation rules are applied as the word is read from left to right. For this problem, you will write a program that chops a word into its respective sounds, substituting any “w” or “k” with “v” or “ts” as appropriate.

**Input Specification:**

Your program will be presented with a sequence of words, one per line. Each word will be a non-empty sequence of lowercase alphabetic characters with perhaps some backwards apostrophes (“’”), always between two vowels. Words will not exceed 75 characters in length.

When your program encounters the string “paupaa”, meaning finished in Hawai‘ian, that is the last word it should process.

**Output Specification:**

On one line at a time, output a modified version of each word, as described above. Illustrate all breaks in sound by inserting single hyphens in the appropriate places.
Sample Input:
aloha
hawai‘i
kauai
napoopoo
paia
ainu
a‘inu
paupaa

Sample Output:
a-lo-ha
ha-vai-i
kau-a-i
na-po-o-po-o
pa-i-a
ai-nu
a-i-nu
pau-pa-a
Problem B- Grilled Cheese

Whenever I make a grilled cheese, it’s the cheater’s way: using cheese slices and English muffins. The strategy? Pre-toast the bread (ostensibly a circle) in the toaster oven, and then, about 90 seconds from the end, stick the cheese slice (ostensibly a square) on top to make some melty melty goodness.

While the result is delicious, there are two problems. As you remember from your course in Properties of Deformable Solids, melted cheese has a lot less structural integrity than solid cheese. In other words, although initially all connected, the cheese furthest off the surface of the bread melts and falls into the oven, lost forever. The other problem is that sometimes the cheese slice doesn’t land centred on the bread.

Input Specification:

The input begins with the positive integer $T \leq 40$, the number of test cases that follow.

Each test case is composed of two lines, the first describes the bread and the second the cheese. The bread is described by the integers $x\ y\ r$, a circle of radius $r > 0$ centred at $(x, y)$. The cheese is described by $x_1\ y_1\ x_2\ y_2$, the coordinates of two corners, such that $|x_2 - x_1| = |y_2 - y_1| > 0$. All input values will never exceed $\pm 1000$, and in every case, the cheese strictly overlaps the centre of the bread, i.e., not just touching an edge or a corner.

Output Specification:

For each test case, output the amount of cheese lost to the oven forever. An iota of cheese will be lost if and only if its location is closer to the original outer edge of the cheese than to the bread. Express your answer as an area with 3 decimals, rounded to the nearest 0.001.

Sample Input:

```
2
0 0 6
6 -6 -6 6
0 0 6
-7 9 -5 -3
```

Sample Output:

```
126.116
109.568
```
Problem C- Hand Swap

Clock puzzles are a fun breed because there are some mathematical results that can, on occasion, be non-intuitive and therefore gratifying. In this puzzle, you are to consider a clock with an hour hand and a minute hand. The hour hand moves at a rate of $30^\circ$ clockwise per hour, whereas the minute hand moves at a rate of $360^\circ$ clockwise per hour.

Every so often, an oddity occurs. If you were to switch the hour hand with the minute hand, it would give you a valid time. Certainly, 12 o’clock is one such time, but 6 o’clock is not. In the case of 6 o’clock, it might look like it is showing a time of 12:30, but if it was really 12:30, the hour hand would be at an angle of 15 degrees - halfway between the 12 and the 1.

Believe it or not there are $N$ unique solutions to this problem. That’s right! $N!!!$

Input Specification:
You will be presented with $N + 1$ lines of input to this problem, each line containing a unique integer between 0 and $N$.

Output Specification:
For each input $k$, output a time on a line by itself, in “HH:MM:SS” format rounded to the nearest second. When $k = 0$ or $k = N$, output “12:00:00”, otherwise, output the $k^{th}$ such time since 12 o’clock where a valid hand swap occurs.

Sample Input:
0
...

Sample Output:
12:00:00
...

---

1 Yes, the problem setter is aware that there is a solution to this problem on the Internet. Remember that part of the challenge of a computer programming contest is to find the solution yourself, or to write a computer program that finds the solution for you.
Problem D- Channel Surfing

My 87 year old dad now has 450 thousand channels like the rest of you youngin’s, but when it comes to remote controls, he is still living in the distant past. No matter how many buttons you give the guy, he only uses 3: “on/off”, “channel up”, and “channel down”.

The **up** and **down** buttons do the obvious thing: move to the next available channel in the sequence. Channel scrolling is in effect, i.e., pressing **up** when on the highest channel moves to the lowest channel number and vice versa for the **down** button. Pressing the **on** button [when the tv was off] will start my dad at channel 2, which is always a channel number.

The guy still likes to channel surf, so why don’t you give him a hand? Given a set of channels he wants to catch a glimpse of, optimize his button clicks so he gets this done the fastest.

**Input Specification:**

Each test case is composed of 3 lines. The first line describes the channels, and begins with the positive integer $C \leq 500$, the number of channel ranges. This is followed by a space-separated sequence of $C$ nonoverlapping channel ranges of the form $m-n$, where $2 \leq m \leq n \leq 999$. A number that is not in any of the ranges means that the channel number is unassigned and therefore not in the channel sequence. The second line begins with the positive integer $N$, and is followed by $N$ distinct assigned channel numbers, which are the channels my dad wants to visit. The third line contains an assigned channel number, the channel my dad is currently on.

Input ends on EOF.

**Output Specification:**

Output the smallest number of button clicks required to visit all of my dad’s favorite channels.

**Sample Input:**

```
5 2-60 110-114 160-199 400-400 600-619
7 44 57 162 163 400 617 618
165
```  

**Sample Output:**

```
50
```
Problem E- Polarity

In the classic 15-puzzle, you slide numbered tiles around a $4 \times 4$ grid until you reach the target configuration. The Polarity puzzle also has 15 sliding tiles, but instead of numbers, the tiles are made from clear polarized plastic, where each tile’s polarity is positive (+) or negative (−). The pieces are held against a $4 \times 4$ square background, also made of clear polarized plastic, and when the tile and the background have opposing polarity, it appears black. So basically, the puzzle’s configuration looks like a $4 \times 4$ array of black and clear squares.

Your program’s job is to make a target configuration of black and clear squares, using the minimum number of moves.

A single move means sliding up to 3 pieces in the same row or the same column as the empty square, with one exception. You can’t manipulate the interior 4 tiles except by gravity. In other words, you may slide the tiles around the outside ring one, two or three at a time, but to move the interior tiles, you must tip the puzzle, thereby sliding all of the tiles in that row or column until the empty square is on the outside ring. A tip counts as one move.

Oh, and as creative as it sounds, you are not allowed to rotate or reflect the puzzle to arrive at a solution.

Input Specification:

There are no more than 100 test cases, presented 10 lines per test case. The first 5 lines are the $4 \times 4$ starting configuration, plus a blank line; the next 5 lines are the $4 \times 4$ finishing configuration, plus a blank line. Each configuration is 4 rows of 4 symbols each, where “x” means the square is black, “.” means the square is clear, and “ ” represents the blank square (which always appears clear). There will be exactly one blank square per starting configuration and exactly no blank squares per finishing configuration.

The background never changes, so it is not included as part of the input. The polarity of the background is shown in the figure.

Input ends on EOF.

Output Specification:

For each test case, print a line with the minimum number of moves required to reach the finishing configuration. If it’s not possible to do so, output “Impossible.”
Sample Input:

....
....
....
....

x..x
....
x..x
.xx.

.x..
x x.
xxx.
x.x.

....
xx..
x.x.
xx..

....
....
....
....

xxxx
xxxx
xxxx
xxxx

Sample Output:

14
7
Impossible.
Problem F- Zurch Trees

In the population of the futuristic world of Zurch, everyone is very logical. City road networks are laid out as trees: every intersection is connected by a unique path of roadway. Crimes are also rare, but when they are committed, it is by a less logical member of society, a Zimeon.

To get away with it, Zimeons need to evade the Zurch Police, who are very adept at detecting and capturing a Zimeon, should they cross each other’s path. The problem is that Zurch Police travel at a slow, logical pace.. almost infinitely slow compared to a Zimeon. Fortunately, the Zurch Police have a reputation of always catching their man, by using their famous Zurch Strategy, a methodically, logical approach to searching a city so that, no matter how fast, clever or lucky a Zimeon is, they will always be caught.

Input Specification:

The input will begin with an integer $N \leq 1000$, the number of test cases. Following that will be $N$ cities, one city per test case. Each city is specified by a positive integer $n \leq 2000$, the number of intersections, followed by $n - 1$ lines describing every connection between a pair of intersections, numbered 1 to $n$.

Output Specification:

Output the minimum number of Zurch Police required to catch even the luckiest Zimeon.

Sample Input:

```
2
2
1 2
8
1 4
2 4
3 4
4 5
5 6
5 7
5 8
```

Map 1:

```
Map 2:
```

Sample Output:

```
1
2
```