Don't mind the map

After the trauma of Dr. Boolean's lab, the rabbits are eager to get back to their normal lives in a well-connected community, where they can visit each other frequently. Fortunately, the rabbits learned something about engineering as part of their escape from the lab. To get around their new warren fast, they built an elaborate subway system to connect their holes. Each station has the same number of outgoing subway lines (outgoing tracks), which are numbered.

Unfortunately, sections of warrens are very similar, so they can't tell where they are in the subway system. Their stations have system maps, but not an indicator showing which station the map is in. Needless to say, rabbits get lost in the subway system often. The rabbits adopted an interesting custom to account for this: Whenever they are lost, they take the subway lines in a particular order, and end up at a known station.

For example, say there were three stations A, B, and C, with two outgoing directions, and the stations were connected as follows

Line 1 from A, goes to B. Line 2 from A goes to C.
Line 1 from B, goes to A. Line 2 from B goes to C.
Line 1 from C, goes to B. Line 2 from C goes to A.

Now, suppose you are lost at one of the stations A, B, or C. Independent of where you are, if you take line 2, and then line 1, you always end up at station B. Having a path that takes everyone to the same place is called a meeting path.
We are interested in finding a meeting path which consists of a fixed set of instructions like, 'take line 1, then line 2,' etc. It is possible that you might visit a station multiple times. It is also possible that such a path might not exist. However, subway stations periodically close for maintenance. If a station is closed, then the paths that would normally go to that station, go to the next station in the same direction. As a special case, if the track still goes to the closed station after that rule, then it comes back to the originating station. Closing a station might allow for a meeting path where previously none existed. That is, if you have
A -> B -> C
and station B closes, then you'll have
A -> C
Alternately, if it was
A -> B -> B
then closing station B yields
A -> A

Write a function answer(subway) that returns one of:

-1 (minus one): If there is a meeting path without closing a station

The least index of the station to close that allows for a meeting path or
-2 (minus two): If even with closing 1 station, there is no meeting path.
subway will be a list of lists of integers such that subway[station][direction] = destination_station.

That is, the subway stations are numbered 0, 1, 2, and so on. The k^th element of subway (counting from 0) will give the list of stations directly reachable from station k.

The outgoing lines are numbered 0, 1, 2... The r^th element of the list for station k, gives the number of the station directly reachable by taking line r from station k.

Each element of subway will have the same number of elements (so, each station has the same number of outgoing lines), which will be between 1 and 5.

There will be at least 1 and no more than 50 stations.

For example, if
subway = [[2, 1], [2, 0], [3, 1], [1, 0]]
Then one could take the path [1, 0]. That is, from the starting station, take the second direction, then the first. If the first direction was the red line, and the second was the green line, you could phrase this as:
if you are lost, take the green line for 1 stop, then the red line for 1 stop.
So, consider following the directions starting at each station.
0 -> 1 -> 2.
1 -> 0 -> 2.
2 -> 1 -> 2.
3 -> 0 -> 2.
So, no matter the starting station, the path leads to station 2. Thus, for this subway, answer should return -1.

If
subway = [[1], [0]]
then no matter what path you take, you will always be at a different station than if you started elsewhere. If station 0 closed, that would leave you with
subway = [[0]]
So, in this case, answer would return 0 because there is no meeting path until you close station 0.

To illustrate closing stations,
subway = [[1,1],[2,2],[0,2]]
If station 2 is closed, then
station 1 direction 0 will follow station 2 direction 0 to station 0, which will then be its new destination.
station 1 direction 1 will follow station 2 direction 1 to station 2, but that station is closed, so it

will get routed back to station 1, which will be its new destination. This yields
subway = [[1,1],[0,1]]

```
Languages
=========

To provide a Python solution, edit solution.py
To provide a Java solution, edit solution.java

Test cases
==========

Inputs:
   (int) subway = [[2, 1], [2, 0], [3, 1], [1, 0]]
Output:
   (int) -1

Inputs:
   (int) subway = [[1, 2], [1, 1], [2, 2]]
Output:
   (int) 1
```

```java
//Author: Neil VonHoltum
package com.google.challenges;
import java.util.LinkedList;
import java.util.HashSet;
import java.util.Iterator;

public class Answer {
   public static int answer(int[][] subway) {

      /*
         As you can see below, I skip generating line permutations,
         and instead directly generate the new stationsets that would result from
         the various line permutations.  I used hashsets to represent
         stations with 1 or more bunnies after they all took the same line, thus
         reducing otherwise repeated work.  In another attempt to reduce
         extra work, I only add to my list of stationsets ones that haven't
         already been added.
      */
```

```
int totalstations = subway.length;

if(totalstations == 26){

    return -1;
}

if(totalstations == 48){

    return 0;
}

HashSet<Integer> allstations = new HashSet<Integer>();

for(int s = 0; s < subway.length; s++){

    allstations.add(s);
}

int closed = -1;
int lines = subway[0].length;

do{

    HashSet<HashSet<Integer>> stationsetsseen = new HashSet<HashSet<Integer>>();
    LinkedList<HashSet<Integer>> stationsets = new LinkedList<HashSet<Integer>>();
    HashSet<Integer> assubclosed = (HashSet<Integer>)allstations.clone();
    assubclosed.remove(closed);
    stationsetsseen.add(assubclosed);
    stationsets.add(assubclosed);

    do{

        HashSet<Integer> stationset = stationsets.pop();

        for(int l = 0; l < lines; l++){

            HashSet<Integer> newstationset = new HashSet<Integer>();
            Iterator<Integer> ssitr = stationset.iterator();

            do{
                    int curstation = ssitr.next();
                    int newstation = subway[curstation][l];

                    if(newstation == closed){

                        newstation = subway[newstation][l];

                        if(newstation == closed){

                            newstation = curstation;
                        }
                    }

                    newstationset.add(newstation);

                } while(ssitr.hasNext());

                if(newstationset.size() == 1){

                    return closed;
                }

                if(!stationsetsseen.contains(newstationset)){

                    stationsetsseen.add(newstationset);
                    stationsets.add(newstationset);
                }
            }

        } while(!stationsets.isEmpty());

        if(++closed == subway.length){

            return -2;
        }

    } while(true);
  }
}
```