Problem Statement

You are arranging a weird game for a team building exercise. In this game there are certain locations that people can stand at, and from each location there are paths that lead to other locations, but there are not necessarily paths that lead directly back. You have everything set up, but you need to know two important numbers. There might be some locations from which every other location can be reached. There might also be locations that can be reached from every other location. You need to know how many of each of these there are.
Create a class TeamBuilder with a method specialLocations that takes a String[] **paths** that describes the way the locations have been connected, and returns a int[] with exactly two elements, the first one is the number of locations that can reach all other locations, and the second one is the number of locations that are reachable by all other locations. Each element of **paths** will be a String containing as many characters as there are elements in **paths**. The *i*-th element of **paths** (beginning with the 0-th element) will contain a '1' (all quotes are for clarity only) in position *j* if there is a path that leads directly from *i* to *j*, and a '0' if there is not a path that leads directly from *i* to *j*.

Definition

Class:	TeamBuilder
Method:	specialLocations
Parameters:	String[]
Returns:	int[]
Method signature:	int[] specialLocations(String[] paths)
(be sure your method is public)	

Constraints

- **paths** will contain between 2 and 50 elements, inclusive.
- Each element of **paths** will contain *N* characters, where *N* is the number of elements of **paths**.
- Each element of **paths** will contain only the characters '0' and '1'.
- The *i*-th element of **paths** will contain a zero in the *i*-th position.

Examples

0)

```
{"010","000","110"}
Returns: { 1,  1 }
```

Locations 0 and 2 can both reach location 1, and location 2 can reach both of the other locations, so we return {1,1}.

1)

```
{"0010","1000","1100","1000"}
Returns: { 1,  3 }
```

Only location 3 is able to reach all of the other locations, but it must take more than one path to reach locations 1 and 2. Locations 0, 1, and 2 are reachable by all other locations. The method returns {1,3}.

2)

```
{"01000","00100","00010","00001","10000"}
```

```
Returns: { 5,  5 }
```

Each location can reach one other, and the last one can reach the first, so all of them can reach all of the others.

3)

```
{"0110000","1000100","0000001","0010000","0110000","1000010","0001000"}
Returns: { 1,  3 }
```