## Problem Statement

It's Sunday morning and you decide to take your girlfriend for a romantic trip. You don't have time to plan anything fancy, so your only option is a trip through a small rectangular forest surrounded by skyscrapers. You know that somewhere, deep inside the forest, there is a flower. You are eager to venture with your girlfriend into the heart of the forest and show her your discovery. Unfortunately, people have left their garbage all over the forest, and you suspect the following about your girlfriend:

- She really hates walking through garbage.

- She doesn't feel very comfortable walking along the garbage either.

You know where all the garbage is, so you want to plan out your path ahead of time. The String[] **forest** contains the map of the forest, where each character represents a cell. 'S' represents your starting point, 'F' represents the flower's location, 'g' represents a cell filled with garbage, and '.' is a clean empty cell. Your goal is to find a path from 'S' to 'F' that contains the fewest possible garbage-filled cells. You may only move one cell at a time, horizontally or vertically. If there are multiple such paths, you want to choose one from among them with the fewest empty cells that have at least one garbage-filled neighbor. Two cells are neighbors if they are horizontally or vertically adjacent. Note that the 'S' and 'F' cells do not count as empty cells.

Return a int[] containing exactly two elements. The first element is the number of garbage-filled cells in the path. The second element is the number of empty cells in the path that have at least one garbage-filled neighbor.

## Definition

| | |
|---|---|
| Class: | ForestGarbage |
| Method: | bestWay |
| Parameters: | String[] |
| Returns: | int[] |
| Method signature: | int[] bestWay(String[] forest) |

(be sure your method is public)

## Constraints

- **forest** will have between 3 and 50 elements, inclusive.
- All elements in **forest** will have the same number of characters.
- Each element in **forest** will have between 3 and 50 characters, inclusive.
- All characters in **forest** will be either 'S', 'F', 'g' or '.'.
- There will be exactly one 'S' in forest, and it will be located on one of the edges.
- There will be exactly one 'F' in forest, and it will not be located on any of the edges.

## Examples

0)

```
{"......",
 "g..F..",
 "......",
 "..g...",
 "......",
 "...S.g"}
```

```
Returns: {0, 0 }
```

Here we can get to the flower without needing to walk through or along garbage. One solution is to take one step up, one step right, three steps up and finally one step left. The picture below describes the forest map with the "maze" created by the garbage, where 'x' denotes a cell that neighbors at least one 'g' cell.

```
xx....
gx.F..
..x...
.xgx..
..x..x
...Sxg
```

1)

```
{"....",
 "..F.",
 "....",
 "..S."}
Returns: {0, 0 }
```

No garbage to worry about.

2)

```
{".....",
 "..F..",
 "..g..",
 ".....",
 "ggggg",
 ".....",
 "..S.."}
Returns: {1, 2 }
```

The garbage wall cannot be avoided, but we can minimize the number of garbage walk-alongs by moving 2 steps towards one of the corners, then going 5 steps straight up and another 2 to reach the flower. There are 2 cells situated along garbage cells on our track; one just before entering the garbage wall and one right after.

3)

```
{"........",
 "g..g...g",
 "........",
 ".g...g..",
 "..g.g..S",
 ".F.g....",
 "......g."}
Returns: {0, 3 }
```

4)

```
{"gggggg",
 "gggFgg",
 "gggggg",
 "gggggg",
 "gggggg",
 "gggggg",
 "gSgggg"}
Returns: {6, 0 }
```

With each move we make, we step in a garbage cell. Taking the shortest path, we visit 6 of these cells.

5)

```
{"S............",
 "gggggggggggg.",
 ".............",
 ".gggggggggggg",
 ".............",
 "gggggggggggg.",
 ".F.g...g...g.",
 ".....g...g..."}
Returns: {0, 54 }
```

No matter how long it takes to deviate from the normal track and walk along the garbage, it's still better than going right through it!