Snake

Time Left:00:21:51

Probably all of us remember a game called Snake, where you have to eat as many apples as you can. As you may have guessed it, your task is to write a program which determines whether our snake wins the game or it ends up eating itself. Snake moves along an infinitely large board. It can turn left, right or go straight. Also, the snake - when going straight - can eat an apple under the condition that it is in front of him. If such a scenario occurs our friend extends onto a cell where the meal took place. The initial snake length is 1.

Input

The first line of the input contains number of tests **t** (t < 1001). Each of the next **t** lines consists of an integer **n** (n < 2401) and **n** characters describing snake's movements. Each character is one of four letters: 'L', 'R', 'F' or 'E'.

- 'L' - snake goes on the field on his left
- 'R' - snake goes right
- 'F' - snake goes on the field in front of him
- 'E' - like 'F' but with eating an apple.

Output

For each test you should print *YES* if the snake survived the current game without dying or otherwise print number of the step in which the snake bites itself.

Example

Input:

Python 3

show

```


def cal(n, movements):
    # Allow faster checking of duplicates
    still_visited = {}

    snake_queue = [] # ordered from tail(first) to head (las
    next_coord = [0, 0] # x, y
    snake_queue.append(tuple(next_coord))
    still_visited[tuple(next_coord)] = False

    direction = 0 # 0, 1, 2, 3 for NESW, clockwise
    for move_no in range(len(movements)):
        check_dup = False
        ## Set Direction ########################
        move = movements[move_no]
        if move == "L":  direction -= 1
        elif move == "R": direction += 1
        #elif move == "F": pass
        #elif move == "E": pass
        direction %= 4

        ### Modify queue ########################
        ### Remove tail
        if move != "E": still_visited[snake_queue.pop(0)] =
        ### Set Next Coord #########
        if direction == 0: next_coord[1] += 1
```

```
2
6 FLERFF
8 EEEELLLL
```

Compile & Test Submit

show

Output:

```
YES
7
```

Results

Case	Status	Input	Output	Expected Output
1	Accepted	2 6 FLERFF 8 EEEELLLL	YES 7	YES 7
2	Accepted	4 1 E 1 L 1 R 1 F	YES YES YES YES	YES YES YES YES
3	Accepted	Hidden	Hidden	Hidden
4	Accepted	Hidden	Hidden	Hidden
5	Accepted	Hidden	Hidden	Hidden

Case	Status	Input	Output	Expected Output
6	Accepted	Hidden	Hidden	Hidden
7	Accepted	Hidden	Hidden	Hidden
8	Accepted	Hidden	Hidden	Hidden
9	Accepted	Hidden	Hidden	Hidden
10	Accepted	Hidden	Hidden	Hidden

show