Instituto Politécnico Nacional

Escuela Superior de Cómputo

DISEÑO DE SOLUCIONES CON PROGRAMACIÓN VORAZ

Ejercicio 08

Análisis de Algoritmos

M. en C. Edgardo Adrián Franco Martínez

Grupo: 3CM3

Fecha: 17 / Mayo / 2018

Alumno:

Calva Hernández José Manuel 2017630201

Índice

Scarecrow

Redacción del ejercicio

Taso owns a very long field. He plans to grow different types of crops in the upcoming growing season. The area, however, is full of crows and Taso fears that they might feed on most of the crops. For this reason, he has decided to place some scarecrows at different locations of the field.

The field can be modeled as a $1 \times N$ grid. Some parts of the field are infertile and that means you cannot grow any crops on them. A scarecrow, when placed on a spot, covers the cell to its immediate left and right along with the cell it is on.

Given the description of the field, what is the minimum number of scarecrows that needs to be placed so that all the useful section of the field is covered? Useful section refers to cells where crops can be grown.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing an integer N ($0 < N < 100$). The next line contains N characters that describe the field. A dot (.) indicates a crop-growing spot and a hash (#) indicates an infertile region.

Output

For each case, output the case number first followed by the number of scarecrows that need to be placed.

Sample Input

```
3
3
.#.
11
...##....##
2
##
```

Sample Output

```
Case 1: 1
Case 2: 3
Case 3: 0
```

Captura de aceptación

20853073	12405 Scarecrow	Accepted	C++	0.000	2018-03-01 16:44:49

Explicación de la solución

Recorreremos cada una de las cadenas siguiendo la siguiente lógica:

- Si ingresan un '#' lo ignoraremos directamente.
- Si ingresan un '.', añadiremos una unidad al resultado, a partir de ello, sabemos que la forma más óptima de colocar ese espantapájaros es cubriendo 3 unidades sin importar qué haya en ellas, por lo tanto, ignoraremos las siguientes dos casillas que nos ingresen prosiguiendo con el algoritmo descrito.

Código

```
#
include < bits / stdc++.h > using namespace std;
int main(int argc, char
    const * argv[]) {
    int t, n;
    cin >> t;
    for (int i = 0; i < t; ++i) {
        cin >> n;
        char c;
        int result = 0;
        while (n-- > 0) {
            cin >> c;
            if (c == '#') {
                continue;
            } else if (c == '.') {
                result++;
                if (n > 2) {
                    n -= 2;
                    cin >> c;
                    cin >> c;
                } else {
                    while (n-- > 0) {
                        cin >> c;
                    }
                }
            }
        }
        cout << result << endl;
    }
    return 0;
}
```

Análisis de complejidad

La complejidad es muy sencilla debido a que únicamente tenemos dos datos importantes, n y t, una vez ingresados, haremos iteraciones recorriéndolos de manera lineal en el peor de los casos, por tanto, la complejidad del problema en general viene dada por:

$$f_t(n,t) = O(nt)$$