



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



-----ANÁLISIS DE ALGORITMOS-----

ACTIVIDAD

Diseño de soluciones mediante Greedy

PROFESOR:

Franco Martínez Edgardo Adrián

ALUMNO:

Meza Vargas Brandon David – 2020630288

GRUPO:

3CM13



índice

Problema: Bear and Row 01	3
Redacción	3
Captura de aceptación por juez	3
Explicación Algoritmo	4
Análisis de complejidad en cota $O()$	5
Código de solución completo	6
Problema: Scarecrow	7
Redacción	7
Captura de aceptación por juez	7
Explicación Algoritmo	7
Análisis de complejidad en cota $O()$	8
Código de solución completo	9

Problema: Bear and Row 01

Redacción

Limak is a little polar bear. He is playing a video game and he needs your help.

There is a row with N cells, each either empty or occupied by a soldier, denoted by '0' and '1' respectively. The goal of the game is to move all soldiers to the right (they should occupy some number of rightmost cells).

The only possible command is choosing a soldier and telling him to move to the right as far as possible. Choosing a soldier takes 1 second, and a soldier moves with the speed of a cell per second. The soldier stops immediately if he is in the last cell of the row or the next cell is already occupied. Limak isn't allowed to choose a soldier that can't move at all (the chosen soldier must move at least one cell to the right).

Limak enjoys this game very much and wants to play as long as possible. In particular, he doesn't start a new command while the previously chosen soldier moves. Can you tell him, how many seconds he can play at most?

Input


The first line of the input contains an integer T denoting the number of test cases. The description of T test cases follows.

The only line of each test case contains a string S describing the row with N cells. Each character is either '0' or '1', denoting an empty cell or a cell with a soldier respectively.

Output

For each test case, output a single line containing one integer — the maximum possible number of seconds Limak will play the game.

Captura de aceptación por juez

ID	Date/Time	Username	Result	Time	Mem	Lang	Solution
54770091	52 sec ago	brand_mv132	 50 (50pts)	0.00	5.3M	C	View

Explicación Algoritmo

```
while(i < N){  
    /*Mientras haya un cero en la cadena  
    while(S[i] == '0' && i < N){  
        i++;  
        /*Si Llegamos al final de la cad  
        if(i == N)  
            break;  
    }  
    /*Cuando nos encontremos con un uno  
    if(S[i] == '1' && i < N)  
        res += seg + seg * (i - j - 1);  
    /*Mientras haya un uno vamos moviend  
    while(S[i] == '1' && i < N){  
        seg++;          /*Aumentan los s  
        i++;            /*Avanzamos la d  
        /*Si Llegamos al final de la cad  
        if(i == N)  
            break;  
    }  
  
    if(i < N)  
        j = i - 1;  
}
```

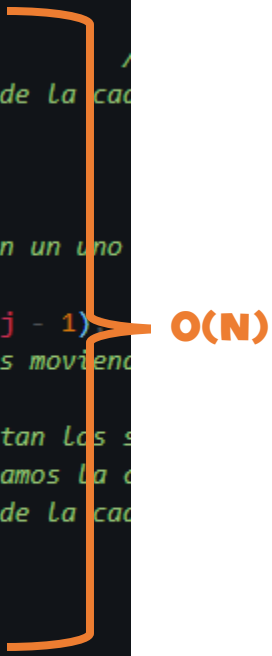
En la captura anterior podemos ver el algoritmo, primeramente se hace un ciclo que va a recorrer toda la cadena de 1's y 0's, posteriormente otro ciclo donde se recorre dicha cadena mientras exista 0 dentro de ella, si se llega al final salimos del ciclo y proseguimos.

Si encontramos un uno, aumentaremos la cantidad de segundos necesarios haciendo una multiplicación de la resta e los índices para ir sumando estos segundos.

Posteriormente vamos recorriendo esos unos a la parte de la izquierda ay por cada movimiento vamos aumentando un segundo hasta que llegamos al final de la cadena o ya no encontremos un 1 en la cadena.

Análisis de complejidad en cota $O()$

```
while(i < N){  
    /*Mientras haya un cero en la cadena  
    while(S[i] == '0' && i < N){  
        i++;  
        /*Si Llegamos al final de la cad  
        if(i == N)  
            break;  
    }  
    /*Cuando nos encontremos con un uno  
    if(S[i] == '1' && i < N)  
        res += seg + seg * (i - j - 1)  
    /*Mientras haya un uno vamos moviend  
    while(S[i] == '1' && i < N){  
        seg++;  
        /*Aumentan los s  
        i++;  
        /*Avanzamos la d  
        /*Si Llegamos al final de la cad  
        if(i == N)  
            break;  
    }  
  
    if(i < N)  
        j = i - 1;  
}
```



La complejidad del algoritmo es lineal, pues se recorre toda la longitud de la cadena en el ciclo y los ciclos internos siguen aumentando el índice de la cadena.

Código de solución completo

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char S[1000000];          /*Variable global para las cadenas

Long Long int bearRow(){

    Long Long int N = strlen(S);    /*Obtenemos la longitud de la cadena
    Long Long int i = 0, j = 0;      /*Variables para loop
    Long Long int res = 0;           /*Variable que almacena el resultado
    Long Long int seg = 0;           /*Variable que guarda los segundos

    /*Recorremos toda la cadena
    while(i < N){
        /*Mientras haya un cero en la cadena y no rebasemos el tamaño de la cadena avanzamos en la cadena
        while(S[i] == '0' && i < N){
            i++;                    /*Avanzamos el índice
            /*Si llegamos al final de la cadena salimos del ciclo
            if(i == N)
                break;
        }
        /*Cuando nos encontremos con un uno vamos sumando los segundos
        if(S[i] == '1' && i < N)
            res += seg + seg * (i - j - 1);
        /*Mientras haya un uno vamos moviéndolo a la derecha aumentando los segundos
        while(S[i] == '1' && i < N){
            seg++;                  /*Aumentan los segundos por cada movimiento
            i++;                    /*Avanzamos la cadena
            /*Si llegamos al final de la cadena salimos del ciclo
            if(i == N)
                break;
        }

        if(i < N)
            j = i - 1;
    }

    if(S[N - 1] == '0')
        res += seg + seg * (i - j - 1);

    return res;
}

```

```

int main()
{
    int T = 0;                /*Variable para el número de casos de prueba
    scanf("%d", &T);          /*Leemos los casos de prueba
    while(T--){               /*Leemos los casos de prueba
        scanf("%s", S);
        printf("%d\n", bearRow()); /*Imprimimos el resultado
    }
    /* code */
    return 0;
}

```

Problema: Scarecrow

Redacción

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.

Captura de aceptación por juez

#	Problem	Verdict	Language	Run Time	Submission Date
27036180	12405 Scarecrow	Accepted	ANSI C	0.000	2021-12-07 23:51:20

Explicación Algoritmo

En la siguiente captura podemos ver el algoritmo empleado para resolver este problema.

Primeramente empezamos con un ciclo que recorrerá nuestro campo simulado con la cadena ingresada.

Si nos encontramos con un. tenemos disponible un espacio para poner un espantapájaros, por lo tanto aumentamos la cantidad de estos, posteriormente como este ocupa dos espacios lo indicamos con un arreglo auxiliar.

Si llegamos al final del campo menos dos (espacio que ocupa) y nos encontramos con otro espacio disponible también ponemos un espantapájaros, aumentando el total de estos.

Por último, si llegamos a la última posición y también tenemos un espacio disponible ponemos un espantapájaros aumentando de igual forma el total de estos.

```
for(i=0; i<N-2; i++)
    /*Si nos encontramos con un . podemos poner un
    /*disponible el espacio, esto lo sabemos si te
    if(campo[i] == '.' && campoAux[i] == 0){
        total++;
        /*Ponemos el espantapa
        campoAux[i+1] = 1; /*Indicamos que el esp
        campoAux[i+2] = 1; /*Como ocupa dos lugar
    }
    /*Si el campo en el tamaño N-2 tenemos otro punto
    if(campo[N-2] == '.' && campoAux[N-2] == 0){
        total++;
        /*Aumentamos la cantidad d
        campoAux[i+1] = 1; /*Indicamos que se ha ocup
    }

    /*Si en la penultima posicion hay un punto y no se
    if(campo[N-1] == '.' && campoAux[N-1] == 0)
        total++;
```

Análisis de complejidad en cota $O()$

```
for(i=0; i<N-2; i++)
    /*Si nos encontramos con un . podemos poner un
    /*disponible el espacio, esto lo sabemos si te
    if(campo[i] == '.' && campoAux[i] == 0){
        total++;
        /*Ponemos el espantapa
        campoAux[i+1] = 1; /*Indicamos que el esp
        campoAux[i+2] = 1; /*Como ocupa dos lugar
    }
    /*Si el campo en el tamaño N-2 tenemos otro punto
    if(campo[N-2] == '.' && campoAux[N-2] == 0){
        total++;
        /*Aumentamos la cantidad d
        campoAux[i+1] = 1; /*Indicamos que se ha ocup
    }

    /*Si en la penultima posicion hay un punto y no se
    if(campo[N-1] == '.' && campoAux[N-1] == 0)
        total++;
```

$O(N)$

Al tener una cadena de tamaño N , la complejidad será lineal.

Código de solución completo

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char *campo;
char *campoAux;

int scarecrow(){
    int i = 0;          /*Variable para loops
    int total = 0;      /*Total de espantapájaros
    int N;              /*Tamaño del campo

    scanf("%d", &N);    /*Leemos el tamaño del campo
    campo = malloc(N*sizeof(char)+1); /*Le asignamos memoria al campo
    scanf("%s", campo); /*Leemos el campo {. #}
    campoAux = calloc(N,sizeof(char)); /*Usamos calloc para asignar memoria iniciando en 0's
    total = 0;          /*Inicializamos el total de espantapajaros

    /*Recorremos el campo hasta dos signos menos
    for(i=0; i<N-2; i++)
        /*Si nos encontramos con un . podemos poner un espantapajaros, ademas debe estar
        /*disponible el espacio, esto lo sabemos si tenemos un 0 en el auxiliar
        if(campo[i] == '.' && campoAux[i] == 0){
            total++;      /*Ponemos el espantapajaros
            campoAux[i+1] = 1; /*Indicamos que el espacio se ha ocupado
            campoAux[i+2] = 1; /*Como ocupa dos lugares ocupamos otro
        }
        /*Si el campo en el tamaño N-2 tenemos otro punto y no se ha ocupado tambien colocamos uno ahi
        if(campo[N-2] == '.' && campoAux[N-2] == 0){
            total++;      /*Aumentamos la cantidad de espantapajaro
            campoAux[i+1] = 1; /*Indicamos que se ha ocupado un espacio
        }

        /*Si en la penultima posicion hay un punto y no se ha ocupado tambien ponemos un espantapajaros
        if(campo[N-1] == '.' && campoAux[N-1] == 0)
            total++;

    printf("%d\n", total); /*Imprimimos el resultado

    free(campo);          /*Liberamos memoria
    free(campoAux);       /*Liberamos memoria
}
```

```
int main(){
    int T = 0;          /*Numero de casos
    scanf("%d", &T);    /*Leemos los casos
    while(T--){
        scarecrow();    /*Ejecutamos la función
    }

    return 0;
}
```