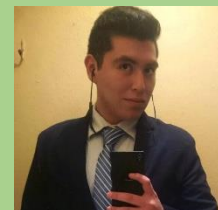
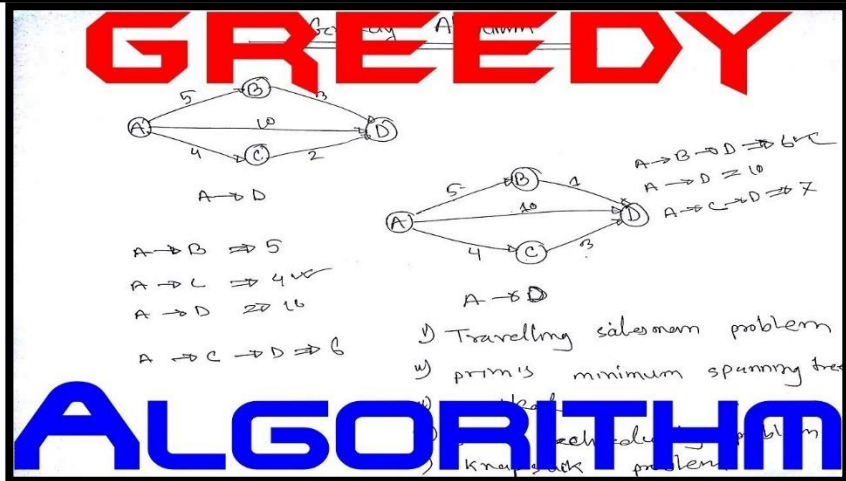


# Análisis de algoritmos



## Diseño de soluciones Greddy

## Bear and row 01

### Descripción

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?

### Entrada

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.

### Salida

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

### Restricciones

- $1 \leq T \leq 5$
- $1 \leq N \leq 10^5$  ( $N$  denotes the length of the string  $S$ )

### Ejemplo

| Entrada         | Salida |
|-----------------|--------|
| 4               |        |
| 10100           | 8      |
| 1100001         | 10     |
| 000000000111    | 0      |
| 001110100011010 | 48     |

### Solución

Para este problema debemos de obtener cierta información para poder hacer el cálculo de forma eficiente, primeramente, debemos iterar sobre toda la cadena, lo primero que debemos hacer es iterar mientras encontremos 0 como carácter, ya que sabemos que los 0 son espacios libres. Vamos a iterar hasta encontrar algún 1, si hemos encontrado un 1 quiere decir que vamos a realizar un desplazamiento, siendo así vamos a hacer el cálculo del desplazamiento y selección. Eso es el número de unos encontrados hasta ahora, mas el número de unos encontrados hasta ahora por la posición actual en la que estamos en la cadena, menos la

posición izquierda del elemento actual, menos uno. Nuevamente nos movemos sobre la cadena hasta encontrar el siguiente 1, si hay más unos iteramos y aumentamos nuestro contador de unos y la posición del elemento en el que estamos “parados” finalmente, si ya se ha recorrido toda la cadena terminamos el ciclo, si salimos del ciclo y aún no se ha recorrido la cadena, entonces j se igualará al valor que hay a la izquierda del carácter actual, y esto se repetirá hasta que se recorra toda la cadena. Una vez que se haya salido del ciclo verificamos si el último carácter que hay en la cadena es un cero, si es así, significa que los unos se tienen que recorrer hasta el final, por tanto, nuevamente calculamos la cantidad de unos obtenida más esa cantidad de unos obtenida por el índice actual del carácter sobre el cual estamos menos la posición del carácter izquierdo menos uno.

## Código

```
#include <bits/stdc++.h>

using namespace std;

typedef long long int lli;
string s;

lli bear_and_row(){

    lli ans = 0, count = 0;
    lli size = s.length();
    lli i = 0, j = 0;
    while(i < size){
        //Moving 'til find a one
        while(s[i] == '0' && i < size){
            i++;
            if(i == size)
                break;
        }

        //We found a one
        if(s[i] == '1' && i < size)
            ans += count + count * (i - j - 1);


        //We move 'til find a zero
        while(s[i] == '1' && i < size){
            //Counting ones
            count++;
            //Moving the index
            i++;
            if(i == size)
                break;
        }
        //We have more elements in the array
        if(i < size)
            j = i - 1;
    }
    if(s[size - 1] == '0')
        ans += count + count * (i - j - 1);
    return ans;
}

int main(){
```

```
    lli T;  
    cin >> T;  
    while(T--){  
        cin >> s;  
        cout << bear_and_row() << endl;  
    }  
    return 0;  
}
```

La complejidad de este algoritmo es lineal, ya que vamos a iterar sobre la cadena una sola vez, ya que los ciclos que hay dentro del ciclo principal mueven nuestra variable  $i$ , la cual es la posición en donde hemos recorrido la cadena. Por tanto podemos decir que el orden  $O$  de el algoritmo es  $O(n)$

### Captura

|          |              |            |   |      |       |       |                      |
|----------|--------------|------------|---|------|-------|-------|----------------------|
| 21657233 | 15 hours ago | eerick1997 |  100<br>[100pts] | 0.01 | 15.7M | C++14 | <a href="#">View</a> |
|----------|--------------|------------|---|------|-------|-------|----------------------|