

# ANÁLISIS DE ALGORITMOS

Alumno: Martín Eduardo Barriga Vargas



Grupo 3CM3

Ejercicio 08: Diseño de soluciones Algoritmos Greedy

26 de mayo de 2019

## Voltea monedas consecutivas

Puntos	20	Límite de memoria	32 MiB
Límite de tiempo (caso)	1s	Límite de tiempo (total)	1m0s
Límite de entrada (bytes)	10 KiB		

### Descripción

Cuentas con un fila de  $N$  monedas, donde algunas de ellas muestran sol y otras muestran aguila, y puedes realizar una operación que voltea un grupo de  $M$  monedas consecutivas. Escribe un programa que calcule la menor cantidad de veces que necesitas realizar la operación para que todas las monedas muestren sol.

### Entrada

Dos enteros  $N$  y  $M$  seguido de  $N$  enteros que corresponden con el estado de las monedas (un 1 denota sol y un 0 denota águila). Puedes suponer que  $1 \leq M \leq N \leq 1000$

### Salida

Un entero que sea la menor cantidad de veces que necesitas realizar la operación o

−1


—

1

si es imposible voltear las monedas.

**Ejemplo**

Entrada	Salida
5 2 1 0 1 0 1	2

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2019-05-23 03:05:08	19773ea8	Respuesta correcta	100.00%	cpp11	3.11 MB	0.02 s	
Nuevo envío							

```

1. #include <iostream>
2. #include <vector>
3. using namespace std;
4.
5. int voltea(vector <int> ar, int n, int m){
6.     int sum = 0;
7.     for(int i = 0; i < n; i++){
8.         if(ar[i] == 0){
9.             if(i + m > n) return -1;
10.            for(int j = 0; j < m; j++ ){

```

```
11.         if(ar[i+j] == 1) ar[i+j] = 0;
12.         else ar[i+j] = 1;
13.     }
14.     sum++;
15. }
16. }
17. return sum;
18.}
19.int main(){
20.    int n,m;
21.    cin >> n >> m;
22.    vector <int> ar(n);
23.    for(int i = 0; i < n; i++){
24.        cin >> ar[i];
25.    }
26.    cout << voltea(ar, n, m) << endl;
27.    return 0;
28.}
```

**Explicación de la solución:**

Se sabe que las monedas que se quieren voltear son las que tienen un 0, pues lo que queremos es terminar con todas las monedas en 1, por lo cual, haremos un recorrido por todas las monedas y cada vez cada que nos encontremos con un 0 vamos a girar esa moneda y las  $m-1$  que se encuentren a su derecha, no sin antes preguntar si el índice de nuestra moneda más  $m$  se sale de la cantidad de monedas que tenemos, en dicho caso entonces regresaremos un -1, indicando que no se pueden voltear todas las monedas, en caso contrario voltearemos los unos por los ceros y los ceros por los unos, además, agregaremos uno a nuestra cantidad de giros que hemos hecho .

Una vez acabado el recorrido, regresaremos la cantidad de giros que tuvimos que hacer.

El algoritmo hace un recorrido dentro de otro, el primero es de tamaño  $n$ , y el de adentro de  $m$ . En el peor de los casos  $m$  será igual a  $n$ , por lo cual la complejidad temporal es  $O(n) = n^2$

## Carrera de larga distancia

Puntos	11.82	Límite de memoria	32 MiB
Límite de tiempo (caso)	1s	Límite de tiempo (total)	1m0s
Límite de entrada (bytes)	10 KiB		

### Descripción

Bessie está entrenando para su próxima carrera corriendo en un camino que incluye colinas de tal manera que ella estará preparada para cualquier terreno. Ella ha planeado un camino recto y quiere correr tan lejos como ella puede, pero ella debe estar de regreso en la granja dentro de **M** segundos.

Todo el camino que ella ha elegido tiene T unidades de longitud y consiste de porciones de la misma longitud de subida, plano o bajada. En los datos de entrada describe un segmento de camino i con un solo carácter S<sub>i</sub> que es U, F, o D, indicando respectivamente subida, plano o bajada.

Bessie gasta U segundos para correr una unidad de camino en subida, F segundos para correr una unidad de camino plano, y D segundos para una unidad de camino en bajada. Note que, cuando regresa a casa, los caminos en subida se vuelven en bajada y los caminos en bajada se vuelven en subida. Encuentre la distancia más lejana a la cual Bessie puede llegar desde la granja y todavía regresar a tiempo.

### Entrada

- Línea 1: Cinco enteros separados por espacios: M, T, U, F, y D
- Líneas 2..T+1: La línea i+1 describe un segmento de camino con una sola letra por renglón.

### Salida

- Línea 1: Un solo entero que es la distancia más lejana (número de unidades) que Bessie puede hacer desde la granja y regresar a tiempo.

### Ejemplo

Entrada	Salida	Descripción
13 5 3 2 1 U F U D F	3	<p>Bessie tiene 13 segundos para regresar a casa (¡trabajo corto!), y el camino total que ella ha planeado tiene 5 unidades de longitud. Ella puede correr una unidad en subida en 3 segundos, una unidad plana en 2 segundos, y una unidad en bajada en 1 segundo. El camino se ve como esto:</p> <pre>       _/\_      / </pre> <p>Ella puede recorrer tres unidades y regresar en <math>3 + 2 + 3 + 1 + 2 + 1 = 12</math> segundos, exactamente un minuto menos que su límite. Si ella intenta ir más lejos, no logrará no sobrepasar el límite.</p>

### Límites

- $1 \leq M \leq 10,000,000$
- $1 \leq T \leq 100,000$
- $1 \leq U \leq 100$
- $1 \leq F \leq 100$
- $1 \leq D \leq 100$

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2019-05-24 00:16:46	<u>18e84013</u>	Respuesta correcta	100.00%	cpp11	3.32 MB	0.08 s	

```
1. #include <iostream>

2. #include <vector>

3. using namespace std;

4. int carrera(vector <char> letras, long long int m, int
    t, int u, int f, int d){

5.     int contador = 0, resta;

6.     for(int i=0; i < letras.size(); i++){

7.         if(letras[i] == 'U' || letras[i] == 'D'){

8.             resta = d+u;

9.         }

10.        else{

11.            resta = f*2;

12.        }

13.        if(m-resta >=0){

14.            m-= resta;

15.            contador++;

16.        }

17.        else{

18.            break;

19.        }
```



```
20.     }
21.     return contador;
22. }
23. int main(){
24.     long long int m;
25.     int t, u, f, d;
26.     char letra;
27.     vector <char> letras;
28.     cin >> m >> t >> u >> f >> d;
29.     for(int i = 0; i < t; i++){
30.         cin >> letra;
31.         letras.push_back(letra);
32.     }
33.     cout << carrera(letras, m, t, u, f, d) <<endl;
34.
35. }
```

**Solución del problema:**

Tenemos nuestras tres variables, que representan el valor de subida, bajada y camino recto. Tenemos que tener presente que una vez que hemos pasado por un metro de alguna de esas tres, debemos de volver por ahí, por lo que cada vez que subamos, nos tardaremos el tiempo de la subida y la bajada por ahí y viceversa, ahora, en caso de que sea recto, nos tardaremos el tiempo de camino recto dos veces.

Sabiendo esto último, hacemos un recorrido por nuestras variables y dependiendo de si subimos/bajamos o tomamos un camino recto, le restaremos el tiempo a nuestra variable de tiempo, además de aumentar el contador de unidades recorridas. Una vez que nuestra variable tiempo sea negativa, romperemos el ciclo y regresaremos el valor del contador final, el cual también regresamos aunque el ciclo haya terminado de forma normal.

Se utiliza un for que recorre nuestro arreglo de tamaño  $n$ , dentro tiene comparaciones de complejidad constante, por lo cual la complejidad de nuestro algoritmo es  $O(n)$

## A - Cake

Tiempo límite: 3 segundos

Memoria límite: 256 megabytes

Lenguajes: MS C# .NET 4.7.2053, GNU G++ 5.1.0 ...

Fito esta intentando construir una montaña de cakes de arroz y para ello tiene varios cakes individuales. El peso del  $i$ -ésimo cake es  $a_i$ , su plan es apilarlos en algún orden tal que satisfaga la siguiente restricción: para cada cake en la pila el peso total de todos los que están encima debe ser estrictamente menor.

Fito necesita tu ayuda para determinar la mayor cantidad de cakes que puede tener la montaña.

### Entrada

La primera línea de la entrada contiene un entero  $N$  ( $1 \leq N \leq 1000$ ). Cada una de las siguientes líneas contiene el peso  $a_i$  del  $i$ -ésimo cake ( $1 \leq a_i \leq 10^9$ )

### Salida

Imprima el mayor tamaño de la montaña de cakes.

### Caso(s) de ejemplo

entrada

5

3

20

5

8

6

salida

3

### Pistas

Por ejemplo, Fito podría apilar los cakes de tamaños 3, 5, 20 de arriba a abajo.

Id	Fecha	Usuario	Problema	Resultado	Tiempo	Memoria	Leng
128067	23/05/2019 1:00:53	martin.barriga.vargas	A - Cake	Accepted	31	421888	GNU G++11 5.1.0

```

1. #include <iostream>

2. #include <algorithm>

3. #include <vector>

4. using namespace std;

5. int cake(vector <int> ar){

6.     int acumulado = 1;

7.     int pesoAcumulado = ar[0];

8.     for(int i = 1; i < ar.size(); i++){

9.

10.         if(pesoAcumulado < ar[i] ){

11.             pesoAcumulado +=ar[i];

12.             acumulado++;

13.         }

14.     }

15.     return acumulado;

```

```
16. }  
  
17. int main(){  
  
18.     int n, entrada;  
  
19.     vector <int> ar;  
  
20.     cin >> n;  
  
21.     for(int i = 0; i < n; i++){  
22.         cin >> entrada;  
23.         ar.push_back(entrada);  
24.     }  
  
25.     sort(ar.begin(), ar.end());  
  
26.     cout << cake(ar) << endl;  
  
27.     return 0;  
  
28. }
```

**Explicación de la solución:**

Como los pasteles no vienen ordenados, lo que haremos será ordenarlos de mayor a menor, luego utilizaremos un ciclo para recorrer nuestro arreglo ya ordenado, empezando por el segundo pastel. Se preguntará si el peso del pastel es mayor que el peso acumulado de los pasteles pasados, si esto es cierto entonces agregaremos su peso al peso acumulado y aumentaremos en uno la cantidad de pasteles apilados. Esto funciona porque lo que siempre queremos hacer será juntar pasteles con la menor cantidad de peso, para que tenga que aguantarlos no tenga que ser muy pesado y el próximo pastel tampoco tenga que pesar tanto.

El algoritmo sólo utiliza un ciclo que recorre el arreglo, y dentro de él sólo hay operaciones de complejidad constante, sin embargo, antes de llamar a la función, se hizo un ordenamiento sort, por lo cual la complejidad final es  $O(n \log n)$

## 12405 Scarecrow

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 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

```
33.#.11...##....##2##
```

Sample Output

Case 1: 1

Case 2: 3

Case 3: 0

#	Problem	Verdict	Language	Run Time	Submission Date
23390860	12405 Scarecrow	Accepted	C++	0.000	2019-05-25 03:31:18

```

1. #include <iostream>

2. #include <vector>

3. using namespace std;

4.

5. int scarecrow(vector <char> ar){

6.     int espantapajaros = 0;

7.     for(int i = 0; i< ar.size(); i++){

8.         if(ar[i] == '.'){

9.             for (int j =i; j < ar.size() && j <= i+2; j++){

10.                 ar[j] = '#';

11.             }

12.             espantapajaros++;

13.         }

```



```
14.     }

15.     return espantapajaros;

16. }

17. int main(){

18.     int casos,tam;

19.     vector <char> ar;

20.     char character;

21.     cin >> casos;

22.     for(int i = 0; i< casos; i++){

23.         cin >> tam;

24.         for(int j = 0; j < tam; j++){

25.             cin >> character;

26.             ar.push_back(character);

27.         }

28.         cout << "Case " << i+1 << ":
    " << scarecrow(ar) << endl;

29.         ar.clear();

30.     }

31.

32.     return 0;
```

33. }

### Explicación de la solución:

La solución es algo parecida a la de las monedas, pues se tiene que hacer un barrido en la cadena y dentro de ese barrido checar en las posiciones siguientes a ella.

Sabemos que forzosamente tenemos que plantar un espantapájaros cada que veamos un punto, ya que es tierra fértil, para esto avanzaremos una unidad y ahí será donde lo plantemos, aprovechando así la tierra que esté en esa unidad y en la siguiente. Para facilitar esto, haremos que esas casillas se vean como tierra infértil colocando # en esas posiciones, así ya no tendremos que preocuparnos por un espantapájaros en esas posiciones, cada vez que hagamos eso sumaremos una unidad en nuestro acumulado de espantapájaros y al final regresaremos ese valor.

La complejidad de nuestro algoritmo es lineal, pues aunque hacemos unas cuantas cosas dentro del for, sabemos que siempre son tres operaciones, quedando así  $O(3n)$

## Bear and Row 01

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.

## Constraints

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

## Subtasks

- Subtask #1 (25 points):  $1 \leq N \leq 10$
- Subtask #2 (25 points):  $1 \leq N \leq 2000$
- Subtask #3 (50 points): Original constraints.

## Example

### Input:

```
4
10100
1100001
000000000111
001110100011010
```

### Output:

```
8
10
0
48
```

Status

Correct Answer

✓

Time

0.01 sec

Solution

24369026

✕

Sub-Task	Task #	Result (time)
1	0	AC (0.000000)
1	1	AC (0.000000)
Subtask Score: 25.00%		Result - AC
2	2	AC (0.000000)
Subtask Score: 25.00%		Result - AC
3	3	AC (0.010000)
3	4	AC (0.010000)
Subtask Score: 50.00%		Result - AC
Total Score = 100.00%		

```

1. #include <iostream>

2. using namespace std;

3. long long int calcular(string s){

4.     long long int acum = 0, i=0, recorrido=0,
       soldado=0;

5.     while(s[i] != '1'){

6.         i++;

```

```
7.     }
8.     i++;
9.     for(; i < s.size(); i++){
10.         while(s[i]!='1' && i < s.size()){
11.             recorrido ++;
12.             i++;
13.         }
14.         acum += soldado*recorrido + recorrido;
15.         if(recorrido > 0) acum+= soldado + 1;
16.         soldado ++;
17.         recorrido = 0;
18.     }
19.     return acum;
20. }

21. int main(){
22.     int n;
23.     string s;
24.     cin >> n;
25.     for (int i = 0; i < n; i++){
26.         cin >> s;
```

```
27.         cout << calcular(s) << endl;  
28.     }  
29. }
```

### Explicación de la solución:

Este problema es un poco más complejo de analizar, lo primero que se debe de hacer es posicionarnos sobre el primer soldado, luego entraremos en un ciclo que recorrerá las posiciones hasta el final, dentro de este ciclo colocaremos otro ciclo en el que deberemos de saber cuantos lugares vacíos recorre el soldado hasta encontrarse con otro soldado o topar con el final del espacio, los lugares que recorra ese soldado también lo tendrán que recorrer todos los soldados anteriores que hemos pasado, por esto tendremos que llevar un registro de soldados que alguna vez hemos visto y dentro de un acumulado sumar la cantidad de espacios que el soldado tuvo que recorrer más la multiplicación de espacios por los soldados anteriores. Además, sumaremos las decisiones que se han tomado, pero sólo si hicimos algún avance con el soldado, en dicho caso sumaremos 1 más las decisiones de los soldados anteriores que tuvimos que mover. Finalmente recorreremos el acumulado.

Este algoritmo tiene una complejidad lineal, pues sólo llevamos un contador y recorremos el string, aunque dentro de ese ciclo tenemos otro ciclo, siempre actualizamos el primero con la posición en el que se quedó el segundo ciclo.

Así  $O(n)$ .