



Instituto Politécnico Nacional  
Escuela Superior de Cómputo  
Análisis de Algoritmos



## Tarea 07

Programacion dinamica

***Alumno:***

López Manríquez Ángel

(2017630941)



M. en C. Edgardo Adrián Franco Martínez

Grupo: 3CM3  
Fecha: 22 de junio de 2019

## Índice

<b>1. Longest Common Subsequence</b>	<b>2</b>
1.1. Descripcion . . . . .	2
1.2. Explicacion . . . . .	2
1.3. Codigo . . . . .	3
1.4. Validacion del juez . . . . .	4
<b>2. El problema de la mochila</b>	<b>4</b>
2.1. Descripción . . . . .	4
2.2. Explicacion . . . . .	4
2.3. Codigo . . . . .	5
2.4. Validacion del juez . . . . .	6
<b>3. Bacterias</b>	<b>6</b>
3.1. Descripción . . . . .	6
3.2. Codigo . . . . .	7
3.3. Explicacion . . . . .	8
3.4. Validacion del juez . . . . .	8

## 1. Longest Common Subsequence

### 1.1. Descripcion

Al finalizar su viaje por cuba, Edgardo se puso a pensar acerca de problemas más interesantes que sus alumnos podrían resolver.

En esta ocasión tu trabajo es el siguiente: Dadas 2 cadenas A y B, debes de encontrar la subsecuencia común más larga entre ambas cadenas.

### 1.2. Explicacion

Leemos las cadenas,  $s_a$  y  $s_b$  representan el tamaño de la cadena  $A$  y  $B$ , respectivamente (necesarias para crear la tabla de Programación Dinámica). Al final se imprime el resultado (entero) de la LCS (enviando como parámetros los arreglos,  $A$  y  $B$ ).

En la función `LCS()` se obtiene la subsecuencia común más larga, usando la recursion hallada en clase. Se usa la funcion sobrecargada `max()` que ya viene definida en C++.

Para crear la tabla, tenemos

- Primero, se crea la tabla que tiene dimensiones de:  $(s_a + 1)(s_b + 1)$ .
- Posteriormente, rellenamos la matriz con ceros (la primera columna y la primera fila deben tener solo ceros).
- Luego, se hace el cálculo, comparando cada posición en B con una posición específica en A, en cada bucle. Debemos tomar en cuenta, que tenemos tres casos:
  1. Si el número es el mismo, tenemos que sumar el número de la parte superior izquierda de la posición actual con 1.
  2. Si no son iguales, tenemos que verificar el máximo entre los números de la posición superior e izquierda de la posición actual.
  3. Si estamos en la primera fila o columna, se debe conservar el cero.
- Finalmente, el resultado debe quedar en la última posición de la tabla, por lo que se retorna su último campo.

Debido a que en la función `LCS()` se tiene que hacer un recorrido a través de la tabla de dimensiones  $s_a s_b$ , tenemos una complejidad de  $O(n^2)$ .

### 1.3. Código

```
1 #include <iostream>
2 #include <string>
3 #include <math>
4
5 #include <stdio.h>
6
7 using namespace std;
8
9 int LCS(const string &A, const string &B) {
10     const int s_a = A.size(), s_b = B.size();
11     int lcs[s_a + 1][s_b + 1] = { { 0 } }; // init everything with 0
12
13     for (int i = 0; i <= s_a; ++i) { // compare each position on B with an specific position on
14         A
15         for (int j = 0; j <= s_b; ++j) {
16             if (i == 0 || j == 0)
17                 lcs[i][j] = 0;
18             else
19                 (A[i - 1] == B[j - 1]) ? (lcs[i][j] = 1 + lcs[i - 1][j - 1]) :
20                     (lcs[i][j] = max(lcs[i - 1][j], lcs[i][j - 1]));
21         }
22     }
23     return lcs[s_a][s_b];
24 }
25
26 int main (void) {
27     string A, B;
28
29     cin >> A;
30     cin >> B;
31
32     printf("%d\n", LCS(A, B));
33 }
```

## 1.4. Validacion del juez

The screenshot shows a judge interface for a sequence matching problem. The input sequence is "AGCT AMGXTP" and the output sequence is "3". The description states: "La subsecuencia comun mas larga es: AGT". The limits are defined as  $1 \leq |A| \leq 10^3$  and  $1 \leq |B| \leq 10^3$ . The source is cited as "Filiberto Fuentes". A new submission was accepted by "galloska".

Submitted	GUID	Status	Percentage	Language	Memory	Runtime	Details
2018-11-22 18:35:00	66f94303	Accepted	100.00%	cpp11	3.41 MB	0.01 s	
New submission							

## 2. El problema de la mochila

### 2.1. Descripción

El famoso problema de la mochila. Empaque para unas vacaciones en el lado del mar y solo llevará una bolsa con capacidad  $S$  ( $1 \leq S \leq 2000$ ). También tiene  $N$  ( $1 \leq N \leq 2000$ ) elementos que puede llevar consigo al lado del mar. Lamentablemente, no puede colocarlos todos en la mochila, por lo que tendrá que elegir. Para cada artículo se le da su tamaño y su valor. Desea maximizar el valor total de todos los artículos que va a traer. ¿Cuál es este valor total máximo?

### 2.2. Explicación

En la función principal, primero necesitamos la cantidad de artículos ( $n$ ) y el tamaño máximo que soporta la mochila ( $s$ ), luego, definimos dos matrices, una para tamaños y otra para valores. Obtenemos el valor y peso de cada artículo y, finalmente, llamamos a la función KS() e imprimimos el resultado, que debe ser entero.

La función KS(), construye la tabla DP de manera ascendente (Bottom up) y calcula cada parte de la tabla. Se requiere como parámetros a los arreglos de tamaños y valores, y la longitud de ambos ( $s, n$ ).

El resultado de la función será el valor máximo del arreglo de valores, de modo que la suma de los pesos (tamaños) de este subconjunto sea menor o igual que  $s$  (peso total de la mochila). Dentro de la función, se define al peso del  $s$ -ésimo artículo como  $j$  (usado en los recorridos). Creada la tabla, se hace un recorrido y en cada bucle, tenemos que verificar los tres casos que pueden suceder de acuerdo con la solución teórica vista en las clases.

- Si estamos en la primera fila o primera columna, ponemos un cero.
- Si el tamaño / peso acumulado es menor que el peso total de la mochila ( $j$ ), Tenemos que contar el valor máximo obtenido entre:
  - valor obtenido por  $i - 1$  artículos y peso (excluyendo el  $n$ th artículo).
  - el valor del  $n$ th artículo más el valor máximo obtenido por  $i - 1$  artículos y  $j$  menos el peso del  $n$ th elemento.
- Si no se cumplen esas condiciones, guardamos el valor obtenido por  $i - 1$  artículos y el peso (Excluyendo el  $n$ th artículo).

Es importante recordar que si el peso del  $n$ -ésimo elemento es mayor que  $W$ , entonces no puede ser incluido. Cuando finalice el bucle, tendremos el resultado en la posición final en la tabla. La complejidad del algoritmo donde se hace la programación dinámica (KS()), será de  $O(nW)$ , siendo  $n$  el número de objetos/elementos y  $W$  el peso total que aguanta la mochila.

### 2.3. Código

```

1 #include <iostream>
2 #include <vector>
3
4 using std::vector;
5 using std::cout;
6 using std::cin;
7
8 /** KS(), this function creates the table of DP (matrix) and computes every part of the table.
9  * */
10 int KS(vector<int> &sizes, vector<int> &values) {
11     int i, j; // j stands for the weight of the sth item.
12     int ks_table[n + 1][s + 1]; // create the DP table
13     const int s = sizes.size(), n = values.size();
14
15     for (i = 0; i <= n; i++) {
16         for (j = 0; j <= s; j++) {
17             if (i == 0 || j == 0) // put a zero if we are in the first row or column
18                 ks_table[i][j] = 0;
19             else if (sizes[i - 1] <= j)
20                 ks_table[i][j] = max(values[i - 1] + ks_table[i - 1][j - sizes[i - 1]],
21                                     ks_table[i - 1][j]);
22             else
23                 ks_table[i][j] = ks_table[i - 1][j];
24         }
25     }
26 }
```

```

21         ks_table[i][j] = ks_table[i - 1][j];
22     }
23 }
24
25 return ks_table[n][s]; // returns the result on the final position at the table.
26 }
27
28 int main (void) {
29     int s, n;
30     cin >> s;
31     cin >> n;
32     vector<int> sizes(s), values(n); // create the quantity of items and the maximum weight
33     ↵ supported by the bag
34     for (int i = 0; i < n; i++) {
35         cin >> sizes[i];
36         cin >> values[i];
37     }
38     cout << KS(sizes,values,s,n);
39     return 0;
}

```

## 2.4. Validacion del juez

The screenshot shows the 'judge status' page of the Sphere online judge. At the top, there are navigation links: PROBLEMS, STATUS, RANKS, DISCUSS, and CONTEST. Below the header, the title 'judge status' is centered. A navigation bar with buttons for '<', 'Previous', '1', '2', '3', '4', '5', 'Next', and '>' is visible. The main content is a table with the following data:

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
22751382	2018-11-23 01:56:07	ang3lino	The Knapsack Problem	accepted edit ideone it	0.00	6.5M	C++ 4.3.2
22751381	2018-11-23 01:55:12	azadesed	The Next Palindrome	wrong answer	6.53	28M	PYTHON3

## 3. Bacterias

### 3.1. Descripción

Un grupo de biólogos computacionales ha diseñado un experimento para decidir si una colonia de microbios es capaz de resolver problemas cuando se le estimula de ciertas formas específicas. En este experimento se construye un recipiente con la forma de una cuadrícula rectangular con  $m$  renglones y  $n$  columnas. En cada uno de los cuadritos de la cuadrícula se coloca cierta cantidad de un compuesto químico que le es muy desagradable a los microbios y que, por lo tanto, los microbios preferirían evitar lo más posible. El recipiente está inclinado de tal forma que los microbios solo puede avanzar hacia el este o hacia el

sur. Por supuesto, los microbios tampoco pueden salir del recipiente. Al principio la colonia de microbios está localizada en el cuadrito correspondiente al primer renglón y primera columna del recipiente. Al final se espera que la colonia de microbios termine en el cuadrito correspondiente al último renglón y última columna del recipiente. En términos de un sistema de coordenadas, los microbios comienzan en la coordenada (1, 1) y terminan en la coordenada (m, n). Antes de llevar a cabo el experimento, los científicos desean calcular la cantidad c de unidades del compuesto químico que deberá soportar la colonia en su trayecto, esto es, la suma de todas las cantidades del compuesto químico que fueron depositadas en todos los cuadritos por los que pasen. En el ejemplo se muestra un recipiente donde el mínimo valor posible de c es 17.

### 3.2. Código

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int m, n;
5
6 void solve(int &grid[m] [] , int &ans[n] []) {
7     for (int i = 0; i < m; ++i) {
8         for (int j = 0; j < n; ++j) {
9             if (i == 0 && j == 0)
10                 ans[i][j] = grid[i][j];
11             else if (j == 0)
12                 ans[i][j] = ans[i - 1][j] + grid[i][j];
13             else if (i == 0)
14                 ans[i][j] = ans[i][j - 1] + grid[i][j];
15             else
16                 ans[i][j] = min(ans[i][j - 1], ans[i - 1][j]) + grid[i][j];
17     }
18 }
19 }
20
21 int main(){
22     cin >> m >> n;
23     int grid[m][n];
24     int ans[m][n] = { { 0 } }; // init it with 0
25     for (int i = 0; i < m; ++i) {
26         for (int j = 0; j < n; ++j) {
27             cin >> grid[i][j];
28         }
29     }
30     solve(grid, ans);
31     cout << ans[m - 1][n - 1] << endl;
32     return 0;
33 }
```

### 3.3. Explicacion

De nuevo usaremos programación dinámica. Sea  $D(i, j)$  el costo mínimo de ir de  $(0, 0)$  a  $(i, j)$  solo moviéndonos abajo y a la derecha. Tenemos solo dos formas posibles de llegar a la casilla  $(i, j)$ :

1. Llegando por la izquierda.
2. Llegando por arriba.

Por lo tanto, el costo mínimo para llegar a  $(i, j)$  es el mínimo de los costos para llegar a  $(i, j - 1)$  o a  $(i - 1, j)$ . Es decir,  $D(i, j) = \min(D(i, j - 1), D(i - 1, j)) + A[i][j]$ . Y como casos base tenemos los costos para llegar a las casillas de la primer fila o columna (las orillas superior e izquierda), que simplemente son la suma de los valores que están a la izquierda o arriba de la casilla incluyéndola, respectivamente. Y obviamente, para llegar a  $(0, 0)$  el costo es simplemente  $A[0][0]$ . Por lo tanto:

$$D(i, j) = \begin{cases} A[i][j] & \text{si } i = j = 0 \\ D(i - 1, j) + A[i][j] & \text{si } i \neq 0 \text{ y } j = 0 \\ D(i, j - 1) + A[i][j] & \text{si } i = 0 \text{ y } j \neq 0 \\ \min(D(i, j - 1), D(i - 1, j)) + A[i][j] & \text{en otro caso} \end{cases}$$

### 3.4. Validacion del juez

Submitted	GUID	Status	Percentage	Language	Memory	Runtime	Details
2018-11-22 19:48:24	45cca45d	Accepted	100.00%	cpp11	3.24 MB	0.00 s	