

## Taller en Sala 10 Programación Dinámica



**Objetivos:** 1. Usar programación dinámica para resolver un problema apropiado. 2. Traducir problemas del mundo real a soluciones algorítmicas.



**Consideraciones:** Lean y verifiquen las consideraciones de entrega,



**Trabajo en Parejas**



**Mañana, plazo de entrega**



**Docente entrega plantilla de código en GitHub**



**Sí .cpp, .py o .java**



**No .zip, .txt, html o .doc**



**Alumnos entregan código sin comprimir GitHub**



En la carpeta Github del curso, hay **un código iniciado** y **un código de pruebas (tests)** que pueden explorar para solucionar los ejercicios



**Estructura del documento:** a) Datos de *vida real*, b) *Introducción* a un problema, c) Problema a resolver, d) Ayudas. Identifiquen esos elementos así:

a)



b)



c)



d)



**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473



En la vida real, la solución de programación dinámica del problema de la subsecuencia común más larga es utilizado en la implementación del comando *diff*, para comparación de archivos, disponible en sistemas Unix. También tiene muchas aplicaciones en bioinformática.

## Ejercicios a resolver

- 1 Actualmente, más del 49% de los desarrolladores de software usan el sistema de control de versiones *git*. Una tarea que hace *git* es determinar, automáticamente, los cambios que se hicieron entre dos versiones de un código fuente.

Para determinar esas diferencias, *git* utiliza el programa *diff*. El programa *diff* resuelve el problema de la subsecuencia común más larga.

```

Viewing: apstring.cpp
/home/joshua/dev/el/cpp/cps111/bigcalc/apstring.h
234 apstring operator + ( const apstring & str, const apstring & str2 ) const
235 // precondition: returns concatenation of str and str2
236 {
237     apstring result; // make string equivalent
238     result = str;
239     result += str2;
240     return result;
241 }
242
243 apstring operator + ( const apstring & str, char ch ) const
244 // precondition: returns concatenation of str and ch
245 {
246     apstring result(str);
247     result += ch;
248     return result;
249 }
250
251
252 apstring apstring::substr(int pos, int len) const
253 //description: extract and return the substring starting at index pos
254 // precondition: this string represents c0, c1, ..., cn-1
255 // 0 <= pos <= pos + len - 1 < n.
256 {
257     // indexing
258     char * operator[] ( int k ) const;
259     char & operator[] ( int k );
260
261     // modifiers
262     const apstring & operator += ( const apstring & str );
263     const apstring & operator += ( char ch );
264
265 private:
266     int mLength; // length
267     int mCapacity; // capacity
268     char * mCString; // storage
269 };
270
271 // The following free (non-member) functions are defined in this file.
272 // I/O functions
273
274

```

El problema de la subsecuencia común más larga es el siguiente: Dadas dos secuencias, encontrar la longitud de la secuencia más larga presente en ambas. Una subsecuencia es una secuencia que aparece en el mismo orden relativo, pero no necesariamente de forma contigua. Para resolver este problema eficientemente, no se puede hacer por *backtracking* porque la complejidad computacional no lo permite.



**Implementen un algoritmo que calcule la longitud de la subsecuencia común más larga a dos cadenas de caracteres utilizando programación dinámica.**

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473



**[Ejercicio Opcional]** Modifiquen el método para retornar, no la longitud de la subsecuencia común más larga, sino la subsecuencia común más larga (es decir, retornar *String* en lugar de *int*).

# Ayudas para resolver los Ejercicios

Ejercicio 1 ..... **Pág. 5**

Ejercicio 2..... **Pág. 5**



## Ejercicio 1



**Ejemplo 1:** “abc”, “abg”, “bdf”, “aeg” y “acefg” son subsecuencias de “abcdefg”. Entonces, para una cadena de longitud  $n$  existen  $2^n$  posibles subsecuencias.

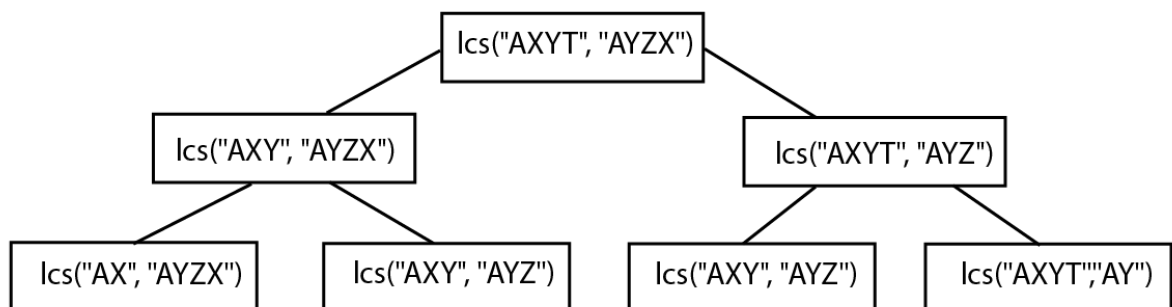


## Ejercicio 2



**Ejemplo 1:** Consideren los siguientes ejemplos para el problema:

Para “ABCDGH” y “AEDFHR” es “ADH” y su longitud es 3. Para “AGGTAB” y “GXTXAYB” es “GTAB” y su longitud es 4. Una forma de resolver este problema es usando *backtracking*, como un ejemplo, para las cadenas “AXYT” y “AYZX”, dada una función recursiva los que resuelve el problema, se obtendría el siguiente árbol (parcial) de recursión:



**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473

## ESTRUCTURA DE DATOS 2

### Código ST0247



**Pista 1:** Usando *backtracking* para ese problema, el problema  $lcs("AXY", "AYZ")$  se resuelve dos veces. Si dibujamos el árbol de recursión completo, veremos que aparecen más y más problemas repetidos, así como en el caso de serie de Fibonacci. Este problema se puede solucionar guardando la soluciones, que ya se han calculado para los subproblemas, en una tabla; es decir, usando programación dinámica.



**Pista 2:** Completen el siguiente método:

```
// Precondición: Ambas cadenas x, y son no vacías
public static int lcsdyn(String x, String y) {
    ...
}
```

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473



# ¿Alguna inquietud?

## CONTACTO

Docente Mauricio Toro Bermúdez

Teléfono: (+57) (4) 261 95 00 Ext. 9473

Correo: mtorobe@eafit.edu.co

Oficina: 19- 627

Agenden una cita dando clic en la pestaña  
-*Semana*- de <http://bit.ly/2gzVg10>