

Laboratorio Nro. 5

Dividi para Conquistar y Programación Dinámica

Objetivos

1. Usar programación dinámica para resolver un problema apropiado
2. Para cada una de las estrategias de diseño de algoritmos, identificar un ejemplo práctico en el que se usaría
3. Usar un algoritmo de dividir y conquistar para resolver un problema adecuado

Consideraciones iniciales

Leer la Guía



Antes de comenzar a resolver el presente laboratorio, leer la **“Guía Metodológica para la realización y entrega de laboratorios de Estructura de Datos y Algoritmos”** que les orientará sobre los requisitos de entrega para este y todos los laboratorios, las rúbricas de calificación, el desarrollo de procedimientos, entre otros aspectos importantes.

Registrar Reclamos



En caso de tener **algún comentario** sobre la nota recibida en este u otro laboratorio, pueden **enviarlo** a través de <http://bit.ly/2q4TTKf>, el cual será atendido en la menor brevedad posible.

Traducción de Ejercicios

En el GitHub del docente, encontrarán la traducción al español de los enunciados de los Ejercicios en Línea.



Visualización de Calificaciones



A través de **Eafit Interactiva** encontrarán **un enlace** que les permitirá **ver un registro de las calificaciones** que **emite el docente** para cada taller de laboratorio y según las rubricas expuestas. **Véase sección 3, numeral 3.7.**

GitHub

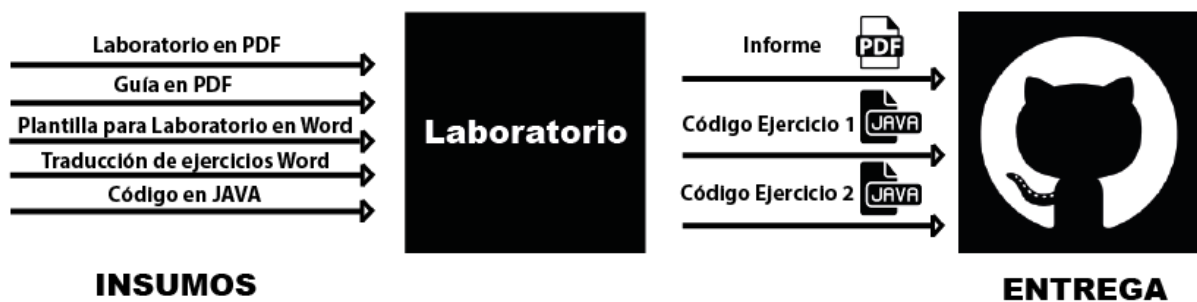


1. Crear un repositorio en su cuenta de GitHub con el nombre `st0247`. 2. Crear una carpeta dentro de ese repositorio con el nombre `laboratorios`. 3. Dentro de la carpeta laboratorio, crear una carpeta con nombre `lab05`. 4. Dentro de la carpeta `lab05`, crear tres carpetas: `informe`, `codigo` y `ejercicioEnLinea`. 5. Subir el informe pdf a la carpeta `infome`, el código del ejercicio 1 a la carpeta `codigo` y el código del ejercicio en línea a la carpeta `ejercicioEnLinea`. Así:

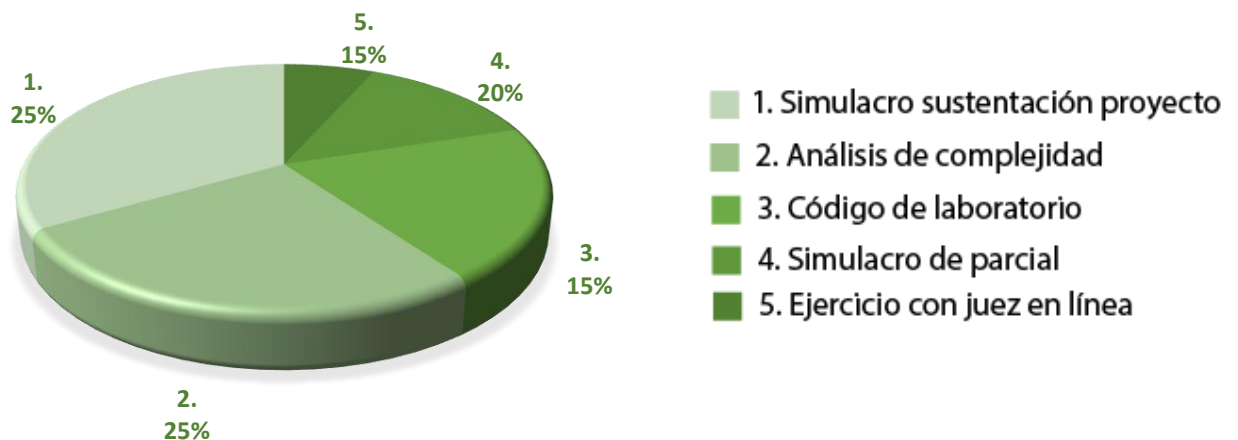
```
st0247
  laboratorios
    lab01
      informe
      codigo
      ejercicioEnLinea
    lab02
      ...
```

Intercambio de archivos

Los archivos que **ustedes deben entregar** al docente son: **un archivo PDF** con el informe de laboratorio usando la plantilla definida, y **dos códigos**, uno con la solución al numeral 1 y otro al numeral 2 del presente. Todo lo anterior se entrega en **GitHub**.



Porcentajes y criterios de evaluación para el laboratorio



DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Resolver Ejercicios

1. Códigos para entregar en GitHub:



En la vida real, la documentación del software hace parte de muchos estándares de calidad como CMMI e ISO/IEC 9126



Véase Guía *en Sección 3, numeral 3.4*



Código de laboratorio en *GitHub*. Véase Guía en *Sección 4, numeral 4.24*



Es opcional entregar documentación. Si lo hace, utilice **Javadoc** o equivalente. No suba el HTML a GitHub.



No se reciben archivos en *.RAR* ni en *.ZIP*



En la vida real, la distancia de Levenshtein se utiliza para algoritmos de reconocimiento óptico de caracteres, es decir, pasar de imagen a texto. También se utiliza en correctores de ortografía, como el que trae Microsoft Word y en especial el de los teclados de los celulares, al igual que en procesamiento del lenguaje natural como Siri de Apple.

1.1 Dadas dos cadenas de caracteres a y b determinen la distancia Levenshtein que hay entre ellas, es decir, la cantidad mínima de operaciones (insertar,

remover o cambiar una letra) que se necesitan para transformar una en la otra utilizando programación dinámica.

Por ejemplo, si tenemos las palabras “carro” y “casa” la distancia Levenshtein que hay entre ellas es 3:

1. Remover una letra: “carr”
2. Cambiar una letra: “casr”
3. Cambiar una letra: “casa”

Nótese que las operaciones y su orden pueden ser diferentes, pero lo importante es que el número mínimo de operaciones para transformar “carro” en “casa” son 3.

Asuman que las cadenas dadas están ambas completamente en minúscula ó mayúscula.



En la vida real, el problema del agente viajero se aplica para la construcción de circuitos electrónicos, recolectar monedas de teléfonos de monedas, entre otras. Léase más en <http://bit.ly/2i9JdIV>

1.2 Implementen el algoritmo de *Held-Karp*, también conocido como algoritmo de programación dinámica para el agente viajero.

Es el mismo problema que hemos trabajado antes (tanto con backtracking como con la técnica *greedy* del vecino más cercano), solo que esta vez lo resolveremos de manera exacta y mediante la forma más eficiente conocida.



Importante: Para que la implementación sea lo más eficiente posible deben representar los conjuntos utilizando máscaras de bits.

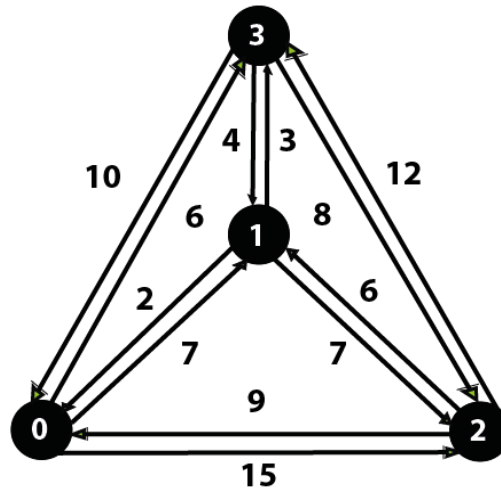
En caso de que tengan problemas con las máscaras de bits puede utilizar la clase *BitmaskSet* que hemos creado para usted, la cual pretende abstraer las

máscaras de bits en forma de *Set* de Java (aunque con muchas limitaciones, pero ideal para la implementación de este algoritmo).

Si deciden usarla recuerden leer la documentación.



Por ejemplo, en el siguiente grafo el costo total del recorrido de costo mínimo que pasa por los cuatro vértices (iniciando y volviendo a 0) de 22:



Iniciando en 0, esta es la tabla de ejecución del algoritmo:

	Costo	Padre
[1, {}]	7	0
[2, {}]	15	0
[3, {}]	6	0
[2, {1}]	14	1
[3, {1}]	10	1
[1, {2}]	21	2
[3, {2}]	27	2
[1, {3}]	10	3
[2, {3}]	14	3
[3, {1, 2}]	$\text{Min } (3 + [1, \{2\}], 12 + [2, \{1\}]) = \text{Min } (24, 26) = 24$	1
[1, {2, 3}]	$\text{Min } (6 + [2, \{3\}], 4 + [3, \{2\}]) = \text{Min } (20, 31) = 20$	2
[2, {1, 3}]	$\text{Min } (7 + [1, \{3\}], 8 + [3, \{1\}]) = \text{Min } (17, 18) = 17$	1
[0, {1, 2, 3}]	$\text{Min } (2 + [1, \{2, 3\}], 9 + [2, \{1, 3\}], 10 + [3, \{1, 2\}]) = 22$	1

Reconstruyendo tenemos $0 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0$.



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.

1.3 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.



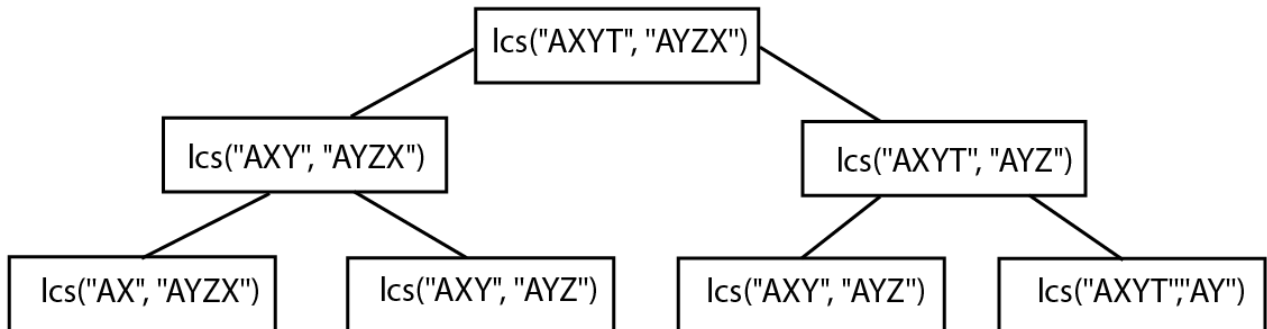
Como un ejemplo, “abc”, “abg”, “bdf”, “aeg” y “acefg” son subsecuencias de “abcdefg”. Entonces, para una cadena de longitud n existen 2^n posibles subsecuencias.

1.4 [Ejercicio Opcional] Modifique el método del ejercicio 1.3 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*)



Como un ejemplo, considere 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 `lcs` que resuelve el problema, se obtendría el siguiente árbol (parcial) de recursión:



2) Ejercicios en línea sin documentación HTML en GitHub:

	Véase Guía en Sección 3, numeral 3.3		No entregar documentación HTML
	Entregar un archivo en .JAVA		No se reciben archivos en .PDF
			Código del ejercicio en línea en GitHub . Véase Guía en Sección 4, numeral 4.24

2.1 Resuelvan el siguiente problema usando programación dinámica:

Karolina es un robot que vive en un sistema rectangular de coordenadas donde cada lugar está designado por un conjunto de coordenadas enteras (x y y).

Su trabajo es diseñar un programa que ayudará a Karolina a escoger un número de desechos radioactivos que están ubicados en su mundo. Para hacerlo, usted debe dirigir a Karolina a la posición en que se encuentra cada uno de los desechos radioactivos.

Encuentre la longitud del camino más corto que llevará a Karolina desde su posición inicial, a cada uno de los desechos radioactivos, y retornar a la posición inicial.

Karolina se puede mover en los ejes x y y , nunca en diagonal, NUNCA. Karolina se puede mover de una posición (i,j) a una posición adyacente $(i,j+1)$, $(i+1,j)$, $(i-1,j)$ o $(i,j-1)$ con costo de 1.

Usted puede asumir que el mundo de Karolina nunca es más grande de 20×20 cuadrados y que nunca habrá más de 10 desechos radioactivos para recoger. Cada coordenada es un par (x,y) donde cada valor está en el rango de 1 al rango máximo de la coordenada respective.

Entrada

Primero, habrá una línea que tiene el número de escenarios que le piden que ayude a Karolina a resolver. Para cada escenario habrá una línea con el tamaño del mundo, dado como dos enteros que representan, respectivamente, el x y el y . Después habrá una línea con dos números que es la posición inicial de Karolina. En la siguiente línea estará el número de desechos radioactivos. Para cada desecho radioactivo, habrá una línea con dos números que representan las coordenadas de cada desecho radioactivo.

Salida

La salida será una sola línea por cada escenario, entregando la distancia mínima que Karolina tiene que moverse para recoger todos los desechos, desde su posición inicial, y regresar, de nuevo, a la posición inicial.

Entrada de ejemplo

```
1
10 10
1 1
4
2 3
5 5
9 4
6 5
```

Salida de ejemplo

The shortest path has length 24

2.2. [Ejercicio Opcional] Resuelvan el siguiente problema <https://goo.gl/bk3zuu>

2.3. [Ejercicio Opcional] Resuelvan el siguiente problema <https://goo.gl/N2LnRY>

2.4. [Ejercicio Opcional] Resuelvan el siguiente problema <https://goo.gl/9jzdx2>

3) Simulacro de preguntas de sustentación de Proyectos



Véase Guía en **Sección 3,**
Numeral 3.4



Entregar informe
laboratorio en **PDF** de



Usen la **plantilla** para
responder laboratorios



**No apliquen Normas
Icontec** para esto

3.1 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 1.2 y cómo funciona el algoritmo.



NOTA: Recuerden que debe explicar su implementación en el informe PDF

3.2 Para resolver el problema del agente viajero, la solución óptima más eficiente que se conoce es la de programación dinámica, desafortunadamente, no es aplicable para un número grande de vértices. Fuera de la solución voraz, ¿qué otros algoritmos aproximados existen?

3.3 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y cómo funciona el algoritmo

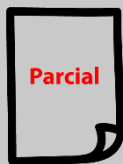


NOTA 1: Recuerden que debe explicar su implementación en el informe PDF

3.4 Calculen la complejidad del ejercicio trabajado en el numeral 2.1 y agréguela al informe PDF

3.5 Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral anterior

4) Simulacro de Parcial en el informe PDF



Para este simulacro, agreguen **sus respuestas** en el informe PDF.

Resuelva, como mínimo, los ejercicios marcados con **color rojo**.



El día del Parcial no tendrán computador, JAVA o acceso a internet.

1. La distancia de Levenshtein es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, una inserción, eliminación o la sustitución de un carácter. Es útil en programas que determinan cuan

similares son dos cadenas de caracteres, como es el caso de los correctores de ortografía.

Como un ejemplo, la distancia de Levenshtein entre "casa" y "calle" es de 3 porque se necesitan al menos tres ediciones elementales para cambiar uno en el otro:

- ☒ Casa a Cala (sustitución de 's' por 'l')
- ☒ Cala a Calla (inserción de 'l' entre 'l' y 'a')
- ☒ Calla a Calle (sustitución de 'a' por 'e')

A continuación esta una implementación del algoritmo en Java:

```
int minimum(int a, int b, int c) {
    return Math.min(Math.min(a, b), c);
}

int Levenshtein(String lhs, String rhs) {
    int[][] distance = new int[lhs.length() + 1][rhs.length() + 1];
    for (int i = 0; i <= lhs.length(); i++)
        distance[i][0] = i;
    for (int j = 1; j <= rhs.length(); j++)
        distance[0][j] = j;
    for (int i = 1; i <= lhs.length(); i++)
        for (int j = 1; j <= rhs.length(); j++)
            distance[i][j] = minimum(distance[i - 1][j] + 1,
                                     distance[i][j - 1] + 1,
                                     ((lhs.charAt(i - 1) == rhs.charAt(j - 1)) ? 0 : 1));
    return distance[lhs.length()][rhs.length()];
}
```

1.1 Complete la siguiente tabla, siguiendo el algoritmo de programación dinámica de la distancia de Levenshtein:

	c	a	l	l	e
c					
a					

s
a

1.2 Complete la siguiente tabla siguiendo el algoritmo de programación dinámica de la distancia de Levenshtein para encontrar la distancia entre madre y mama:

m a d r e
m
a
m
a

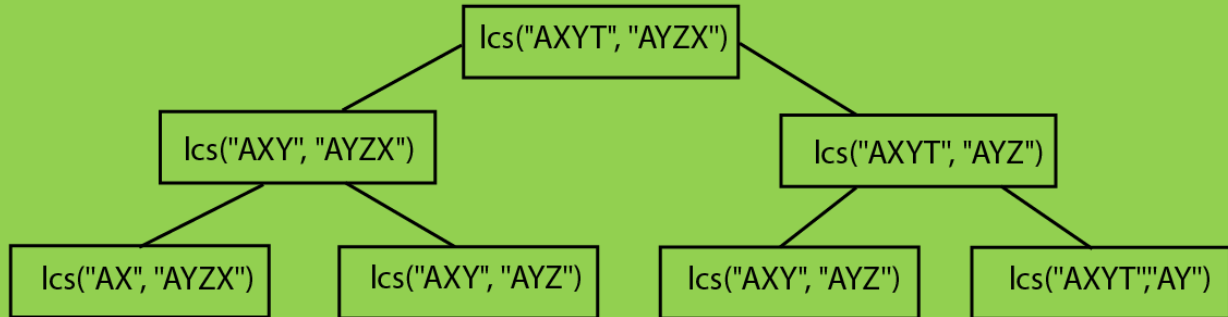
2. El problema de la subsecuencia común más larga es el siguiente: Dadas dos secuencias, encontrar la longitud de la secuencia más larga presente en ambas. Una subsecuencia es una secuencia que aparece en el mismo orden relativo, pero no necesariamente de forma contigua.

Como un ejemplo, “abc”, “abg”, “bdf”, “aeg” y “acefg” son subsecuencias de “abcdefg”. Entonces, para una cadena de longitud n existen 2^n posibles subsecuencias.

Este problema es utilizado en la implementación del comando diff, para comparación de archivos, disponible en sistemas Unix. También tiene muchas aplicaciones en bioinformática.

Considere 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 lcs que resuelve el problema, se obtendría el siguiente árbol (parcial) de recursión:



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, como en el algoritmo siguiente:

```

01 // Precondición: Ambas cadenas x, y son no vacías
02 public static int lcsdyn(String x, String y) {
03     int i,j;
04     int lenx = x.length();
05     int leny = y.length();
06     int[][] table = new int[lenx+1][leny+1];
07
08     // Inicializa la tabla para guardar los prefijos
09     // Esta inicialización para las cadenas vacías
10     for (i=0; i<=lenx; i++)
11         table[i][0] = 0;
12     for (i=0; i<=leny; i++)
13         table[0][i] = 0;
14
15     // Llena cada valor de arriba a abajo
16     // y de izquierda a derecha
17     for (i = 1; i<=lenx; i++) {
18         for (j = 1; j<=leny; j++) {
19             // Si el último caracter es igual
20             if (x.charAt(i-1) == y.charAt(j-1))
21                 table[i][j] = 1+table[i-1][j-1];
22             // De lo contrario, tome el máximo

```

```
23         else // de los adyacentes
24             table[i][j] = Math.max(table[i][j-1],table[i-
1][j]);
25         System.out.print(table[i][j]+" ");
26     }
27     System.out.println();
28 }
29
30 // Retornar la respuesta (tipo entero, ojo)
31
32 }
```

2.1 ¿Cuál es la complejidad asintótica para el peor de los casos? OJO, n no es una variable en este problema.

2.2 Complete la línea 31

3. La serie de Fibonacci esta dada por 1,1,2,3,5,8,13,... La complejidad de calcular el element n -esimo de la serie de Fibonacci, usando el siguiente algoritmo, es...

```
public int fibo(int n) {
    int[] fibos = new int[n+1];
    for (int i = 0; i <= n; i++) {
        if (i<=1)
            fibos[i] = i;
        else
            fibos[i] = fibos[i-1]+fibos[i-2];
    }
    return fibos[n];
}
```

3.1

- a) $O(n)$
- b) $O(n^2)$
- c) $O(2^n)$
- d) $O(3^n)$

e) $O(n!)$

3.2 Porque ...

- a) $T(n) = c_1:n + c_2$
- b) $T(n) = c_1:n^2 + c_2:n$
- c) $T(n) = c_12n + c_2$
- d) $T(n) = c_13n + c_2:n$
- e) $T(n) = c_13n + c_2:n!$

4. La complejidad de calcular el elemento n -ésimo de la serie de Fibonacci, usando el siguiente algoritmo, es... y una forma de optimizar el algoritmo es...

```
public int fibo(int n) {  
    if (n <= 1) return n;  
    else return fibo(n-1)+fibo(n-2);  
}
```

- a) $O(n)$ y se optimiza con programación estática
- b) $O(n^2)$ y se optimiza con programación dinámica
- c) $O(2^n)$ y se optimiza con programación dinámica
- d) $O(3^n)$ y se optimiza con programación dinámica
- e) Ninguna de las anteriores

5. Considere una secuencia de números $a : a_1, a_2, a_3, \dots, a_n$. Para esta secuencia siempre se cumple que $a_1 \leq a_2 \leq \dots \leq a_n$, es decir, esta secuencia está ordenada de menor a mayor (siendo a_1 el menor elemento de a y a_n el mayor elemento de a).

Ahora, considere un entero z . Queremos encontrar, en la secuencia, el mayor numero que es menor o igual a z . La solución a este problema simplemente es hacer **búsqueda binaria** en la secuencia, teniendo en cuenta, en la búsqueda, que el elemento actual sea menor o igual al elemento examinado. Ayúdanos a terminar el siguiente código.

- **Ejemplo:** Sea $a = \{3, 7, 11, 12, 23, 24, 27, 77, 178\}$ y $z = 45$. La respuesta es 27.

```
01 int bus(int[]a,int iz,int de,int z){  
02     if(iz>de){  
03         return -1;    }  
04     if(a[de]>=z){  
05         return a[de]; }
```



```
06     int mitad=(iz+de)/2;
07     if(a[mitad]==z){
08         return .....; }
09     if(mitad>0){
10         if(a[mitad-1]<=z && z<a[mitad]){
11             return mitad - 1;        }    }
12     if(z<a[mitad]){
13         return bus(a,iz,mitad-1,z); }
14     //else
15     return bus(....., ....., ....., .....);
16 }
17 public int bus(int[] a, int z) {
18     return bus(a, 0, a.length - 1, z);
19 }
```

5.1 ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior?

- (a) $T(n)=2.T(n/2)+C$ que es $O(n)$
- (b) $T(n)=2.T(n/2)+Cn$ que es $O(n.\log n)$
- (c) $T(n)=T(n/2)+C$ que es $O(\log n)$
- (d) $T(n)=4.T(n/2)+C$ que es $O(n^2)$

5.2 (10%) Complete la línea 8

5.3 (10%) Complete la línea 15,,,

6. Dado un conjunto de enteros $a=a_1, a_2, \dots, a_n$, queremos encontrar su

subsecuencia creciente máxima. La subsecuencia creciente máxima de una secuencia, es una subsecuencia de esta secuencia cuyos elementos están ordenados y esta subsecuencia es tan larga como sea posible. Los elementos de esta subsecuencia no necesariamente tienen que ser contiguos en la secuencia original. Este problema tiene aplicaciones en bioinformática para análisis de nicho ecológico y para filogenética. **Nota:** En la otra página hay algunos ejemplos.

En este problema estamos interesados en encontrar el tamaño de la subsecuencia creciente máxima. La ecuación recursiva está definida de esta forma:

$$scm(i) = \begin{cases} 1 + \max(scm(j)) \text{ donde } 0 < j < i \text{ y } a_j < a_i; \\ 1 & \text{sino existe tal valor } j \end{cases}$$

El siguiente código hace el trabajo, utilizando programación dinámica, pero faltan algunas líneas, complétalas por favor.

```
01 int scm(int arr[]) {
02     int n = arr.length;
03     int scm[] = new int[n];
04     int i,j,max = 0;
05     /* Inicializar la tabla scm */
06     for ( i = 0; i < n; i++ ) {
07         .....; }
08     /* Calcular usando la tabla scm*/
09     for ( i = 1; i < n; i++ ) {
10         for ( j = 0; j < i; j++ ) {
11             if ( arr[i] > arr[j] && scm[i] < scm[j] + 1) {
12                 .....; } } }
13     /* El maximo en la tabla scm */
14     for ( i = 0; i < n; i++ ) {
15         if ( max < scm[i] ) {
16             .....; } }
17     return max;
```

- **Ejemplo 1:** Sea $a = \{7, 4, 8, 1, 3, 9, 2, 6, 10\}$. La subsecuencia creciente máxima es **$\{1, 2, 6, 10\}$** . Nota que **$\{1, 3, 6, 10\}$** es otra subsecuencia creciente máxima. En todo caso, la longitud de la secuencia creciente máxima es 4.
- **Ejemplo 2:** Sea $a = \{4, 2, 1, 9, 3, 4, 10, 2, 1\}$. La subsecuencia creciente máxima es **$\{1, 3, 4, 10\}$** . La longitud de la secuencia creciente máxima es 4.

6.1 (10%) Complete la línea 7

6.2 (10%) Complete la línea 12

6.3 (10%) Complete la línea 16

6.4 (10%) ¿Cuál es la complejidad asintótica, en el peor de los casos, del método anterior?

- (a) $O(1)$
- (b) $O(n)$
- (c) $O(n^2)$
- (d) $O(n \log n)$

7. El **algoritmo de Floyd-Warshall** encuentra la distancia mínima entre cada par de vértices i, j de un grafo con pesos. Este algoritmo muchas veces es usado para determinar la clausura transitiva de un grafo, que es útil en Lenguajes Formales y

Compiladores. Dada la representación usando **matrices de adyacencia** g de un grafo, donde la posición i,j de la matriz es ∞ si el vértice i no está conectado con el vértice j o la posición i,j es un entero $a_{i,j}$ si existe una conexión entre el vértice i y j , queremos encontrar la distancia mínima entre el vértice u y el vértice v . Ayúdanos a completar el siguiente código:

```
int calcular(int[][] g, int u, int v){
    int n = g.length;
    int[][] d = new int[n][n];
    for(int i = 0; i < n; ++i){
        for(int j = 0; j < n; ++j){
            d[i][j] = g[i][j];
        }
    }
    for(int k = 0; k < n; ++k){
        for(int i = 0; i < n; ++i){
            for(int j = 0; j < n; ++j){
                int ni = .....;
                int nj = .....;
                int nk = .....;
                int res = Math.min(ni, nj + nk);
                d[i][j] = res;
            }
        }
    }
    return d[u][v];
}
```

7.1 Completa la línea 12

7.2 Completa la línea 13

7.3 Completa la línea 14

7.4 ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior?

5. [Ejercicio Opcional] Lectura recomendada



"Quienes se preparan para el ejercicio de una profesión requieren la adquisición de competencias que necesariamente se sustentan en procesos comunicativos. Así cuando se entrevista a un ingeniero recién egresado para un empleo, una buena parte de sus posibilidades radica en su capacidad de comunicación; pero se ha observado que esta es una de sus principales debilidades..."

Tomado de <http://bit.ly/2gJKzJD>



Véase Guía en **Sección 3, numeral 3.5 y 4.20** de la Guía Metodológica, "Lectura recomendada" y "Ejemplo para realización de actividades de las Lecturas Recomendadas", respectivamente

Posterior a la lectura del "**R.C.T Lee et al., Introducción al análisis y diseño de Algoritmos. Capítulo 7: Programación Dinámica.**", realicen las siguientes actividades que les permitirán sumar puntos adicionales:

- a) Escriban un resumen de la lectura que tenga una longitud de 100 a 150 palabras
- b) Hagan un mapa conceptual que destaque los principales elementos teóricos.



NOTA 1: Si desean una lectura adicional en español, consideren la siguiente: "**John Hopcroft et al., Estructuras de Datos y Algoritmos, Sección 10.3. 1983**", que encuentran en biblioteca.



NOTA 2: Estas respuestas también deben incluirlas en el informe PDF

6. [Ejercicio Opcional] Trabajo en Equipo y Progreso Gradual



El trabajo en equipo es una exigencia actual del mercado. "Mientras algunos medios retratan la programación como un trabajo solitario, la realidad es que requiere de mucha comunicación y trabajo con otros. Si trabajas para una compañía, serás parte de un equipo de desarrollo y esperarán que te comuniques y trabajes bien con otras personas"

Tomado de <http://bit.ly/1B6hUDp>



Véase Guía en **Sección 3, numeral 3.6** y **Sección 4, numerales 4.21, 4.22 y 4.23** de la Guía Metodológica

- a) Entreguen copia de todas las actas de reunión usando el tablero Kanban, con fecha, hora e integrantes que participaron
- b) Entreguen el reporte de *git*, *svn* o *mercurial* con los cambios en el código y quién hizo cada cambio, con fecha, hora e integrantes que participaron
- c) Entreguen el reporte de cambios del informe de laboratorio que se genera *Google docs* o herramientas similares



NOTA 1: Estas respuestas también deben incluirlas en el informe PDF

7. [Ejercicio Opcional] Laboratorio en inglés



El inglés es un idioma muy importante en la Ingeniería de Sistemas porque la mayoría de los avances en tecnología se publican en este idioma y la traducción, usualmente se demora un tiempo y es sólo un resumen de la información original.


Adicionalmente, dominar el inglés permite conseguir trabajos en el exterior que son muy bien remunerados

Tomado de goo.gl/4s3LmZ

Entreguen el código y el informe traducido al inglés. Utilicen la plantilla dispuesta en este idioma para el laboratorio

Resumen de ejercicios a resolver

- 1.1 Dadas dos cadenas de caracteres *a* y *b* determinen la distancia *Levenshtein* que hay entre ellas
- 1.2 Implementen el algoritmo de *Held-Karp*, también conocido como algoritmo de programación dinámica para el agente viajero.
- 1.3 Dadas dos secuencias, encontrar la longitud de la secuencia más larga presente en ambas.
- 1.4 [Ejercicio Opcional] Modifiquen el método del ejercicio 1.3 para retornar, no la longitud de la subsecuencia común más larga, sino la subsecuencia común más larga
- 2.2. Resolver el problema de Karolina
- 2.2. Resuelvan el siguiente problema <https://goo.gl/bk3zuu> [Ejercicio Opcional]
- 2.3. Resuelvan el siguiente problema <https://goo.gl/N2LnRY> [Ejercicio Opcional]

	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Cód. ST0247
		Estructuras de Datos 2

2.4. Resuelvan el siguiente problema <https://goo.gl/9jzdx2> [Ejercicio Opcional]

3.1 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 1.2 y cómo funciona el algoritmo.

3.2. Fuera de la solución voraz para resolver el problema del agente viajero, ¿qué otros algoritmos aproximados existen?

3.3 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y cómo funciona el algoritmo

3.4 Calculen la complejidad del ejercicio trabajado en el numeral 2.1 y agréguela al informe PDF

3.5 Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral anterior

4. Simulacro de Parcial

5. Lectura recomendada [Ejercicio Opcional]

6. Trabajo en Equipo y Proceso Gradual [Ejercicio Opcional]

7. Entreguen el código y el informe traducido al inglés. [Ejercicio Opcional]

Ayudas para resolver los ejercicios

Ayudas para el Ejercicio 1.....	<u>Pág. 22</u>
Ayudas para el Ejercicio 1.1.....	<u>Pág. 22</u>
Ayudas para el Ejercicio 1.2.....	<u>Pág. 22</u>
Ayudas para el Ejercicio 1.4.....	<u>Pág. 23</u>
Ayudas para el Ejercicio 3.2.....	<u>Pág. 23</u>
Ayudas para el Ejercicio 3.4.....	<u>Pág. 23</u>
Ayudas para el Ejercicio 4.0.....	<u>Pág. 24</u>
Ayudas para el Ejercicio 5A.....	<u>Pág. 24</u>
Ayudas para el Ejercicio 5B.....	<u>Pág. 24</u>
Ayudas para el Ejercicio 6A.....	<u>Pág. 24</u>
Ayudas para el Ejercicio 6B.....	<u>Pág. 25</u>
Ayudas para el Ejercicio 6C.....	<u>Pág. 25</u>

Ayudas para el Ejercicio 1



PISTA 1: Si deciden hacer la documentación, consulten la *Guía en Sección 4, numeral 4.1* “Cómo escribir la documentación HTML de un código usando Javadoc”

Ayudas para el Ejercicio 1.1



PISTA 1:

```
public static int levenshtein(String a, String b) {  
    // complete...  
}
```



PISTA 2: Como recomendación: solucionen el siguiente problema para tener una mayor seguridad de que su implementación es correcta: <http://www.spoj.com/problems/EDIST/>



PISTA 3: Utilicen el siguiente simulador <http://www.let.rug.nl/kleiweg/lev/>

Ayudas para el Ejercicio 1.2



PISTA 1:

```
public static int heldKarp(Digraph g) {  
    // complete...  
}
```



PISTA 2: Si tienen problemas para la implementación del algoritmo le recomendamos el siguiente video: <http://bit.ly/2kqN9dz> . Recuerden activar los subtítulos (en inglés) en caso de que los necesite.

Ayudas para el Ejercicio 1.4



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) {
    ...
}
```

Ayudas para el Ejercicio 3.2



PISTA 1: Léase <http://bit.ly/2xjXZs1>

Ayudas para el Ejercicio 3.4



PISTA 1: Véase *Guía en Sección 4, numeral 4.11 "Cómo escribir la complejidad de un ejercicio en línea"*

Ayudas para el Ejercicio 4.0



PISTA 1: Véase *Guía en Sección 4, Numeral 4.18* “Respuestas del Quiz”



PISTA 2: Lean las diapositivas tituladas “*Data Structures II: Dynamic programming*”, encontrarán la mayoría de las respuestas

Ayudas para el Ejercicio 5a



PISTA 1: En el siguiente enlace, unos consejos de cómo hacer un buen resumen <http://bit.ly/2knU3Pv>



PISTA 2: [Aquí](#) les explican cómo contar el número de palabras en Microsoft Word

Ayudas para el Ejercicio 5b



PISTA 1: Para que hagan el mapa conceptual se recomiendan herramientas como las que encuentran en <https://cacoo.com/> o <https://www.mindmup.com/#m:new-a-1437527273469>

Ayudas para el Ejercicio 6a



PISTA 1: Véase *Guía en Sección 4, Numeral 4.21* “Ejemplo de cómo hacer actas de trabajo en equipo usando Tablero Kanban”

Ayudas para el Ejercicio 6b



PISTA 1: Véase Guía en Sección 4, Numeral 4.23 “**Cómo generar el historial de cambios en el código de un repositorio que está en svn**”

Ayudas para el Ejercicio 6c



PISTA 1: Véase Guía en Sección 4, Numeral 4.22 “**Cómo ver el historial de revisión de un archivo en Google Docs**”