Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos, Sem: 2021-1, 3CV1, Práctica 7, 23-12-2020

Práctica 7: Programación dinámica

De la Cruz Sierra Diana Paola
Raya Chávez Samuel Antonio
dianapaodcs@gmail.com, samiraya1323@gmail.com

Resumen. Se seguirán viendo problemas que impliquen el uso de la programación dinámica, esta vez enfocada en el problema de subsecuencia común más larga, y de líneas de producción.
Palabras clave. Programación dinámica, optimización, tablas, c++.

1. Introducción

La programación dinámica tiene como finalidad encontrar una solución de un problema de optimización en forma secuencial. La programación dinámica no es un algoritmo de solución única, sino más bien un método para resolver un problema grande y único solventando una secuencia de problemas más pequeños, sin importar el número de ellos.

La solución de problemas mediante la programación dinámica se basa en el llamado principio de optimalidad, el cual, fue enunciado por Bellman (1957) y dice que: "En una secuencia de decisiones óptima toda subsecuencia ha de ser también óptima". Sin embargo, se debe evaluar en cada problema presentado que este principio se esté cumpliendo y analizar cómo abordar cada uno de los problemas de acuerdo a sus propias características.

Algo a tomar en cuenta es que si necesitamos resolver el mismo problema más tarde, podemos obtener la solución de la lista de soluciones calculadas y reutilizarla, pero si estamos seguro de que no la volveremos a utilizar se puede descartar para ahorrar espacio.

La programación dinámica puede tomar uno de dos enfoques llamados top-down y bottom-up

- Top-down: el problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente.
- Bottom-up: los problemas que pudieran ser necesarios se resuelven anticipadamente y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones.

viendo lo anterior puede notarse que debido a que bottom-up no utiliza llamadas a funciones es más rápido.

Además puede no solo se quiera conocer el óptimo, si no también como llegar a ese valor.Por esto, en cada paso se toman decisiones sobre como dividir al problema en subproblemas cuyas soluciones darán la solución óptima y guardar en cada estado toda información necesaria sobre la decisión óptima.

Memoización: Ésta es usada para el enfoque top-down y su función es almacenar resultados de funciones, evitando repetir cálculos para entradas previamente procesadas.

2. Conceptos básicos

- **Programación dinámica.** Los pasos a seguir para poder implementar la programación dinámica son:

- Buscar la forma de que el problema se pueda resolver de una forma óptima.
- Definir que valor se debe de repetir recursivamente.
- Guardar las soluciones obtenidas en cada iteración dentro de una tabla para posteriormente poder consultar los resultados ya obtenidos.
- Construir la solución óptima con base en las tablas ya obtenidas previamente.

Siguiendo estos pasos, se puede obtener una solución óptima siguiendo los principios de la programación dinámica. (Benoso, B., 2020).

- **Problema de la subsecuencia común más larga:** LCS (por sus siglas en inglés Longest Common Subsequence) es un algoritmo que sirve para hallar la subsecuencia común más larga entre dos cadenas de caracteres. (Gaston, 2016).
- **Problema de líneas de producción.** Se busca encontrar la ruta que tome el menor tiempo posible de inicio a fin entre 2 líneas de producción, siguiendo los tiempos de inicio a fin definidos en cada estación de trabajo, si se quiere cambiar de línea entonces puede haber un costo extra de tiempo, todo depende de como estén definidos los tiempos. (Benoso, B., 2020).

2.1. Subsecuencia común más larga

En el algoritmo 1 se muestra el procedimiento para calcular encontrar la subsecuencia más larga dadas dos cadenas de texto X y Y.

Algorithm 1 Pseudocódigo: LCSlength

```
1: procedure LCSLENGTH(String X, String Y)
2:     m ← X.length
3:     m ← Ylength
4:     let b[1..m, 1..n] and c[0..m, 0..n] be new tables
5:     for i ← 1, m do
6:         c[i, 0] ← 0
7:     end for
8:     for j ← 0, n do
9:         c[0, j] ← 0
10:    end for
11:    for i ← 1, m do
12:        for j ← 1, n do
13:            if X[i] == y[j] then
14:                c[i, j] ← c[i − 1, j − 1] + 1
15:                b[i, j] = " ↖ "
16:            else if c[i − 1, j] ≥ c[i, j − 1] then
17:                c[i, j] ← c[i − 1, j]
18:                b[i, j] = " ↑ "
19:            else
20:                c[i, j] ← c[i, j − 1]
21:                b[i, j] = " ← "
22:            end if
23:        end for
24:    end for
25:    return c and b
26: end procedure
```

En el algoritmo 1 si ambas secuencias tienen un símbolo en común, se le suma un 1 al valor que se encuentra en su vecino superior izquierdo, se coloca en la posición actual y se marca con " ↖ " indicando de donde se obtuvo la actualización del valor. Por otro lado si el valor del vecino superior es ≥ a su vecino izquierdo, se copia el valor que tiene su vecino superior y se coloca " ↑ ". En caso contrario si el valor del vecino izquierdo es mayor que el vecino superior, se copia el valor del vecino izquierdo y se marca con " ← ".

Posteriormente en el algoritmo 2 se muestran los pasos para poder imprimir la cadena Z que es la subsecuencia común más larga de las cadenas de texto originales X y Y.

Algorithm 2 Pseudocódigo: printLCS

```
1: procedure PrintLCS(b, X, i, j)
2:     if i == 0 or j == 0 then
3:         return
4:     end if
5:     if b[i, j] == " ↖ " then
6:         printLCS(b, X, i − 1, j − 1)
7:         print X[i]
8:     else if b[i, j] == " ↑ " then
9:         printLCS(b, X, i − 1, j)
10:    else
11:        printLCS(b, X, i, j − 1)
12:    end if
13: end procedure
```

2.2. Líneas de producción

En el siguiente algoritmo se muestra la manera en que de dos líneas de producción se elige el camino más corto para llegar de inicio al fin.

Algorithm 3 Pseudocódigo: Línea producción

```
1: procedure PROCESO(n,a[1,2][1,...,n],t[1,2][1,...,n-1],e[1,2],x[1.2])
2:     f[1][1] ← e[1] + a[1][1]
3:     f[2][1] ← e[2] + a[2][1]
4:     for j = 2, n do
5:         if f[1][j − 1] + a[1][j] ≤ f[2][j − 1] + t[2][j − 1] + a[1][j] then
6:             f[1][j] ← f[1][j − 1] + a[1][j]
7:             I[1][j] ← 1
8:         else
9:             f[1][j] ← f[2][j − 1] + t[2][j − 1] + a[1][j]
10:            I[1][j] ← 2
11:        end if
12:        if f[2][j − 1] + a[2][j] ≤ f[1][j − 1] + t[1][j − 1] + a[2][j] then
13:            f[2][j] ← f[2][j − 1] + a[2][j]
14:            I[2][j] ← 2
15:        else
16:            f[2][j] ← f[1][j − 1] + t[1][j − 1] + a[2][j]
17:            I[2][j] ← 1
18:        end if
19:    end for
20:    if f[1][n] + x[1] ≤ f[2][n] + x[2] then
21:        f∗ ← f[1][n] + x[1]
22:        I∗ ← 1
23:    else
24:        f∗ ← f[2][n] + x[2]
25:        I∗ ← 2
26:    end if
27: end procedure
```

Del algoritmo 3, éste se encargará de almacenar los tiempos que se toman al seleccionar la siguiente estación de trabajo, a la vez que se registrarán en otra tabla unos valores que determinarán cual es la ruta óptima.

Si la suma de las rutas anteriores con respecto a la línea 1 es menor que la suma anterior a la de la línea 2, entonces en una nueva tabla I* se marcara la celda correspondiente a la estación de trabajo actual como 1, indicando que la ruta de la línea es la más corta; pero si la suma de la línea 2 es menor que la de la línea 1, entonces en la celda actual de I* se marcará un 2.

Esta tabla I* servirá como referencia para posteriormente imprimir el camino más óptimo.

Algorithm 4 Pseudocódigo: Imprimir resultados

```
1: procedure RESULTADOS(n,I[1,2][2,...,n],I*)
2:     i ← I*
3:     print"línea" i, "estación" n
4:     for j ← n,downto2 do
5:         i ← I[i][j]
6:         print"línea" i, "estación" j − 1
7:     end for
8: end procedure
```

Es aquí cuando se toma la tabla obtenida previamente del algoritmo 3 para ir recorriendo la tabla I* y obtener la ruta óptima, imprimiendo la línea y la estación a seguir.

3. Experimentación y resultados

3.1. Subsecuencia común más larga

Vamos a comenzar, para el algoritmo 1, generando dos pruebas de escritorio, dadas dos cadenas de texto X como cadena de texto original y Y como una cadena de texto de prueba para conocer su subsecuencia común más larga y que tanto parecido tiene la cadena prueba de la cadena original. Siendo la original del 100 %.

- Ejemplo 1:

  Para las cadenas de texto:

  X = ABCBDAB, m=7

  y = BDCABA, n=6

  comenzamos llenando la tabla C en las posiciones c[i,0]=0 y c[0,j]=0. Esto se aprencia en la tabla 8

	j	**0**	**1**	**2**	**3**	**4**	**5**	**6**
i		**Yj**	**B**	**D**	**C**	**A**	**B**	**A**
0	**Xi**	0	0	0	0	0	0	0
1	**A**	0						
2	**B**	0						
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

Cuadro 1: paso 1: tabla c. Llenamos c[i,0]=0 y c[0,j==0

posteriormente llenamos toda la tabla 8 quedandonos de la siguiente manera: Para la tabla c se tiene:

	j	**0**	**1**	**2**	**3**	**4**	**5**	**6**
i		**Yj**	**B**	**D**	**C**	**A**	**B**	**A**
0	**Xi**	0	0	0	0	0	0	0
1	**A**	0	0	0	0	1	1	1
2	**B**	0	1	1	1	1	2	2
3	**C**	0	1	1	2	2	2	2
4	**B**	0	1	1	2	2	3	3
5	**D**	0	1	2	2	2	3	3
6	**A**	0	1	2	2	3	3	4
7	**B**	0	1	2	2	3	4	4

Cuadro 2: tabla C llena

Y para la tabla b llena se muestra en la tabla 3.

	j	0	1	2	3	4	5	6
i		**Yj**	**B**	**D**	**C**	**A**	**B**	**A**
0	**Xi**	0	0	0	0	0	0	0
1	**A**	0	↑	↑	↑	↖	←	↖
2	**B**	0	↖	←	←	↑	↖	←
3	**C**	0	↑	↑	↖	←	↑	↑
4	**B**	0	↖	↑	↑	↑	↖	←
5	**D**	0	↑	↖	↑	↑	↑	↑
6	**A**	0	↑	↑	↑	↖	↑	↖
7	**B**	0	↖	↑	↑	↑	↖	↑

Cuadro 3: Tabla b llena

Una vez llenadas las tablas se usa el algoritmo 2 y se pasa como paramétro la tabla 3 para imprimir de manera recursiva el camino generado en la tabla 4, pero unicamente el marcado con las flechas ↖.

	j	0	1	2	3	4	5	6
i		**Yj**	**B**	**D**	**C**	**A**	**B**	**A**
0	**Xi**	0	0	0	0	0	0	0
1	**A**	0						
2	**B**	0	↖	←				
3	**C**	0			↖	←		
4	**B**	0					↖	
5	**D**	0					↑	
6	**A**	0						↖
7	**B**	0						↑

Cuadro 4: Dirección LCS

Es decir el resultado es: $Z =$ "**BCBA** ".

Teniendo en cuenta que la cadena original $|X| = 7$ y que $|Z| = 4$, entonces tienen un parecido del 57,14 %

- Ejemplo 2:

  Ahora vamos a desarrollar un ejemplo para:

  X = Çuadro", m=6.

  Y = "Marco", n=5.

comenzamos llenando la tabla C en las posiciones c[i,0]=0 y c[0,j]=0. Esto se aprencia en la tabla 5

	j	**0**	**1**	**2**	**3**	**4**	**5**
i		**Yj**	**M**	**a**	**r**	**c**	**o**
0	**Xi**	0	0	0	0	0	0
1	**C**	0					
2	**u**	0					
3	**a**	0					
4	**d**	0					
5	**r**	0					
6	**o**	0					

Cuadro 5: Ejemplo2. Paso1, tabla c

posteriormente llenamos toda la tabla 5 quedandonos de la siguiente manera: Para la tabla C del ejemplo 2 se tiene:

	j	**0**	**1**	**2**	**3**	**4**	**5**
i		**Yj**	**M**	**a**	**r**	**c**	**o**
0	**Xi**	0	0	0	0	0	0
1	**C**	0	0	0	0	0	0
2	**u**	0	0	0	0	0	0
3	**a**	0	0	1	1	1	1
4	**d**	0	0	1	1	1	1
5	**r**	0	0	1	2	2	2
6	**o**	0	0	1	2	2	3

Cuadro 6: Tabla c. Ejemplo 2

Y para la tabla b llena se muestra en la tabla 7.

	j	0	1	2	3	4	5
i		**Yj**	**M**	**a**	**r**	**c**	**o**
0	**Xi**						
1	**C**		↑	↑	↑	↑	↑
2	**u**		↑	↑	↑	↑	↑
3	**a**		↑	↖	←	←	←
4	**d**		↑	↑	↑	↑	↑
5	**r**		↑	↑	↖	←	←
6	**o**		↑	↑	↑	↑	↖

Cuadro 7: Tabla b. Ejemplo 2

Una vez completas las 2 tablas 6 y 7 se manda a imprimir la cadena dando como resultado: $Z =$ "aro". Cuya longitud es de 3.

Teniendo en cuenta que la longitud de la cadena de texto $|X| = 6$ y que $|Z| = 3$, entonces su parecido es del 50 %

Ahora vamos a mostrar 5 ejemplos de entradas y salidas del algoritmo 1 programado.

- Ejemplo 1:

  Texto original: hola este es el ejemplo 1 de la practica LCS

  Texto prueba: Hi es un ejemplo de la practica siete

  En la imagen 1 se observa que la similitud es del 60 % para el ejemplo 1.

```
C:\Users\Diana Paola\Documents\5to Semestre\Algoritmos\practica7>g++ -o LCS LCS.cpp

C:\Users\Diana Paola\Documents\5to Semestre\Algoritmos\practica7>LCS
Nombre del archivo original: ejemplo1A
Nombre del archivo prueba: ejemplo1B

LCS: esejemplodelapractica

similar: 60.00%
```

Figura 1: Salida y similitud para el ejemplo 1.

- Ejemplo 2:

  Texto original: Cuadro

  Texto prueba: Marco

En la imagen 2 se observa que la similitud es del 50 % para el ejemplo 2.

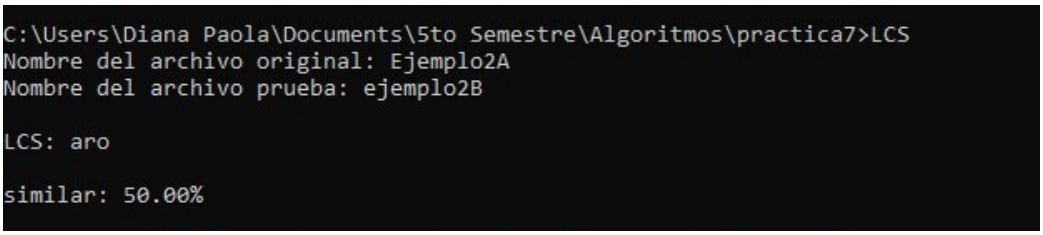


Figura 2: Salida y similitud para el ejemplo 2.

- Ejemplo 3:

  Texto original: En informática, la programación dinámica es un método para reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas, como se describe a continuación. El matemático Richard Bellman inventó la programación dinámica en 1953 que se utiliza para optimizar problemas complejos que pueden ser discretizados y secuencializados.

  Texto prueba: La programación dinámica es un método de optimización que puede aplicarse a diferentes y numerosos problemas, algunos de los cuales ya han sido analizados en programación lineal y programación entera. Los parámetros usados en la programación dinámica pueden ser estocásticos o probabilísticos y determinísticos.

  En la imagen 3 se observa que la similitud es del 45,38 % para el ejemplo 3.

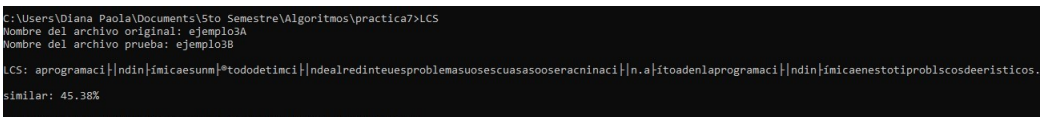


Figura 3: Salida y similitud para el ejemplo 3.

- Ejemplo 4:

  Texto original: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the

1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Texto prueba: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

En la imagen 4 se observa que la similitud es del 100 % para el ejemplo 4, ya que el texto el exactamente el mismo.

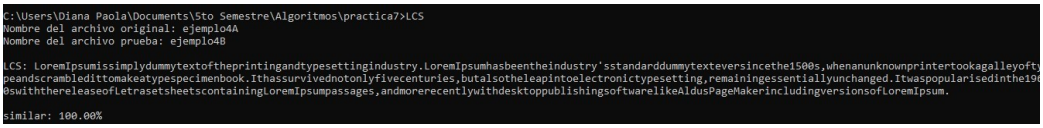


Figura 4: Salida y similitud para el ejemplo 4.

- Ejemplo 5:

  Texto original: ABCBDAB

  Texto prueba: BDCABA

  En la imagen 5 se observa que la similitud es del 57,14 % para el ejemplo 5.

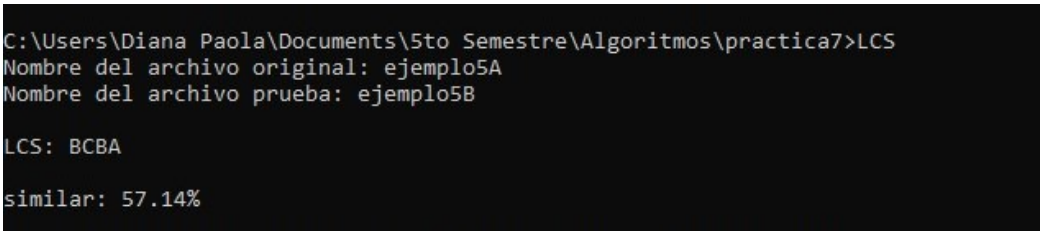


Figura 5: Salida y similitud para el ejemplo 5.

3.2. Líneas de producción

Viendo algunos ejemplos, se tomara el siguiente caso:

- Ejemplo 1

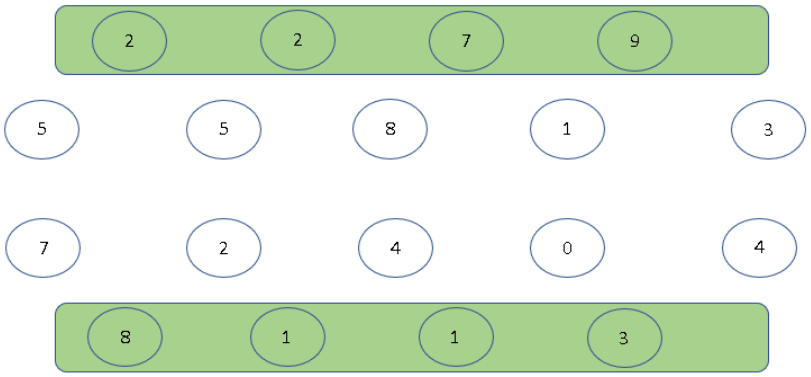


Figura 6: Ejemplo 1: Líneas de producción

Primero se suman los tiempos de entradas con su respectiva línea de producción como se muestra en la figura 7

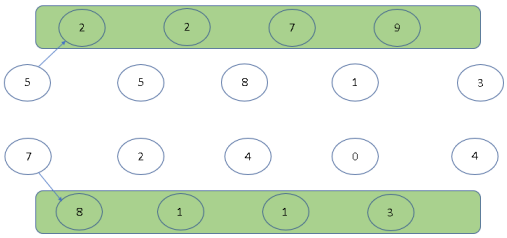


Figura 7: Ejemplo 1, paso 1

Y las tablas quedan de esta forma:

j	**1**	**2**	**3**	**4**
$f_1[n]$	**7**			
$f_2[n]$	15			

j	**2**	**3**	**4**
$I_1[j]$			
$I_2[j]$			

Cuadro 8: Ejemplo 1, paso 1. Selección de la primera celda

La primera tabla se llena con la suma de las entradas y la primera estación de trabajo, en este caso se escoge el 7 ya que es el menor. Para el segundo paso, se suma el 7 obtenido previamente con los nuevos tiempos para los caminos de la línea de producción 1 y la línea de producción 2, como se muestra en la figura 8

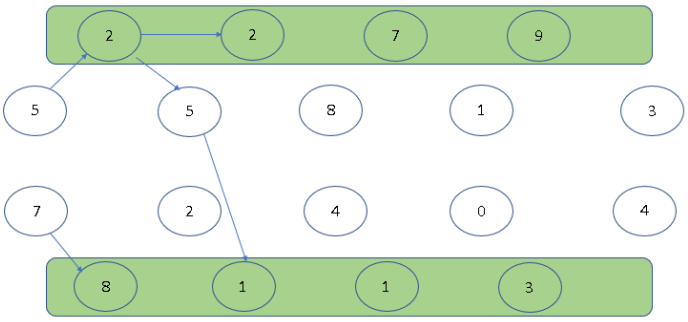


Figura 8: Ejemplo 1, paso 2

Y las tablas quedan de esta forma:

j	**1**	**2**	**3**	**4**
$f_1[n]$	**7**	**9**		
$f_2[n]$	15	13		

j	**2**	**3**	**4**
$I_1[j]$	1		
$I_2[j]$	1		

Cuadro 9: Ejemplo 1, paso 2

Se escoge la menor suma que es 13, y para la celda correspondiente a $f_1[n]$ se suma con 7+2+7 ya que resulta ser menor que sumar 13+8+7; y para $f_2[n]$ se suma la suma obtenida previamente (13) con el 1 correspondiente a la misma línea de producción.

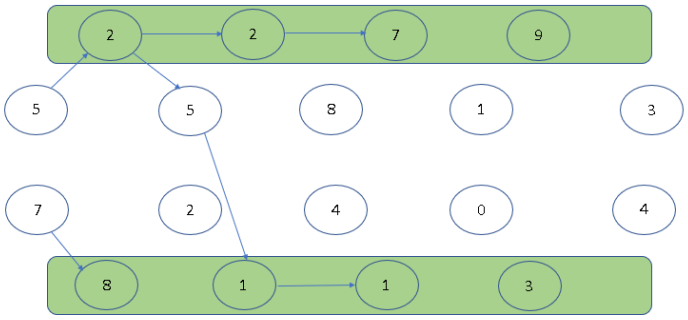


Figura 9: Ejemplo 1, paso 3

Obteniendo las siguientes tablas:

j	1	2	3	4
$f_1[n]$	**7**	**9**	16	
$f_2[n]$	15	13	**14**	

j	2	3	4
$I_1[j]$	1	1	
$I_2[j]$	1	2	

Cuadro 10: Ejemplo 1, paso 3

De la tabla 10, la tabla I pasará de 1 a 2 ya que indica que se empezará a tomar la línea 2 como suma de referencia.

Para $f_1[n]$ se toma la suma 16+0+9, y para $f_2[n]$ se toma 14+3, como se muestra en la siguiente figura.

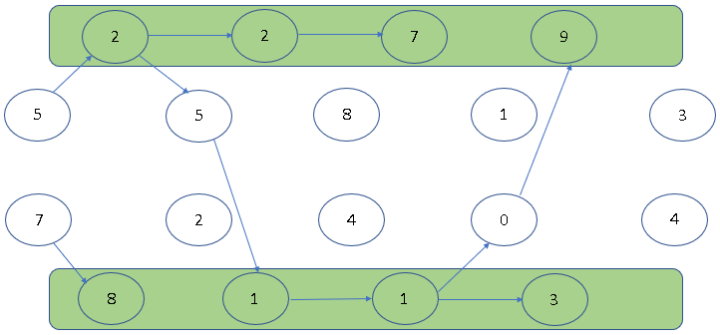


Figura 10: Ejemplo 1, paso 4

Y de las tablas se quedan los datos como:

j	1	2	3	4
$f_1[n]$	**7**	**9**	16	23
$f_2[n]$	15	13	**14**	**17**

j	2	3	4
$I_1[j]$	1	1	2
$I_2[j]$	1	2	2

Cuadro 11: Ejemplo 1, paso 4

Como paso final, se suman las últimas estaciones de trabajo con su salida y así es como en la tabla I se ve reflejada la línea que se tomó, es decir, aquella ruta que tomó el menor tiempo.

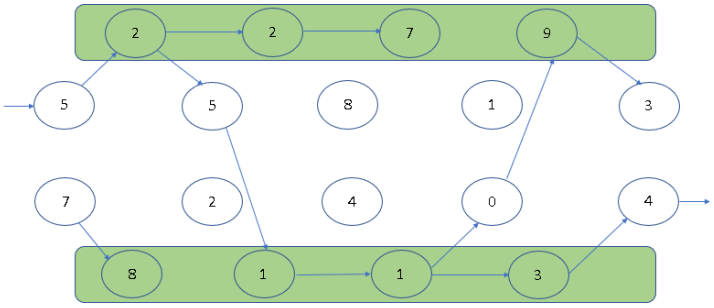


Figura 11: Ejemplo 1, paso 5

Y se obtienen las siguientes tablas:

j	**1**	**2**	**3**	**4**	
$f_1[n]$	**7**	**9**	16	23	26
$f_2[n]$	15	13	**14**	**17**	**21**

j	**2**	**3**	**4**	
$I_1[j]$	1	1	2	**2**
$I_2[j]$	1	2	2	

Cuadro 12: Ejemplo 1, paso 5

De la última tabla se obtiene la ruta óptima siguiendo la indicación de la celda del fin al principio, es decir, si dice 2 se va a la columna anterior y se va a la celda 2 y así sucesivamente, como lo muestra la tabla 13

j	**2**	**3**	**4**	
$I_1[j]$	1	1	2	***2***
$I_2[j]$	***1***	***2***	***2***	

Cuadro 13: Ejemplo 1, paso 6

Es decir, seleccionará la ruta 2→2→2→1
Con el algoritmo 4 se obtienen los siguientes resultados:

- Línea 2, estación 4
- Línea 2, estación 3
- Línea 2, estación 2
- Línea 1, estación 1

Y de esta forma se obtuvo la ruta óptima para estas líneas de producción.
Considerando un nuevo ejemplo, ahora con 7 estaciones de trabajo y omitiendo las demostraciones ya que se seguirán los mismos pasos que con el ejemplo 1, se tiene:

- Ejemplo 2

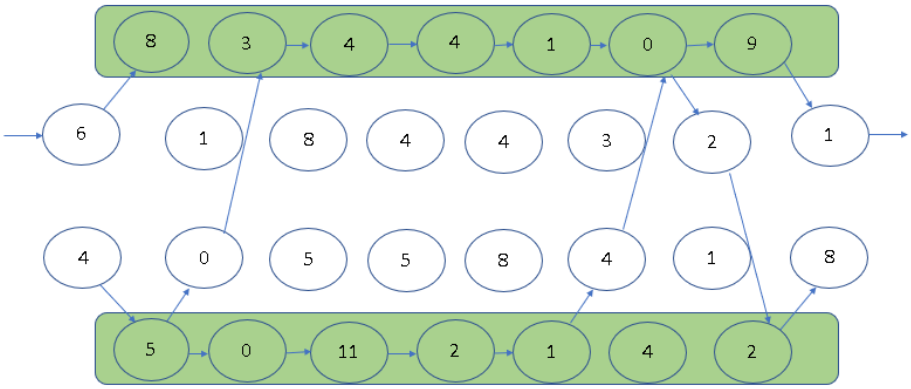


Figura 12: Ejemplo 2

Y las tablas quedan como:

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	
$f_1[n]$	14	12	16	20	21	21	30	31
$f_2[n]$	9	9	20	22	23	27	25	33

j	**2**	**3**	**4**	**5**	**6**	**7**	
$I_1[j]$	2	1	1	1	1	1	1
$I_2[j]$	2	2	2	2	2	2	

Cuadro 14: Ejemplo 2, registro de tablas

Entonces se sigue la ruta 1→1→1→1→1→1→2, y con ayuda del algoritmo 4 se obtiene el siguiente resultado:

- Línea 1, estación 7
- Línea 1, estación 6
- Línea 1, estación 5
- Línea 1, estación 4
- Línea 1, estación 3
- Línea 1, estación 2
- Línea 2, estación 1

Ejemplificando los resultados que se obtuvieron de los ejemplos anteriores con la implementación del código y omitiendo las peticiones de la máquina al usuario, se tienen los resultados:

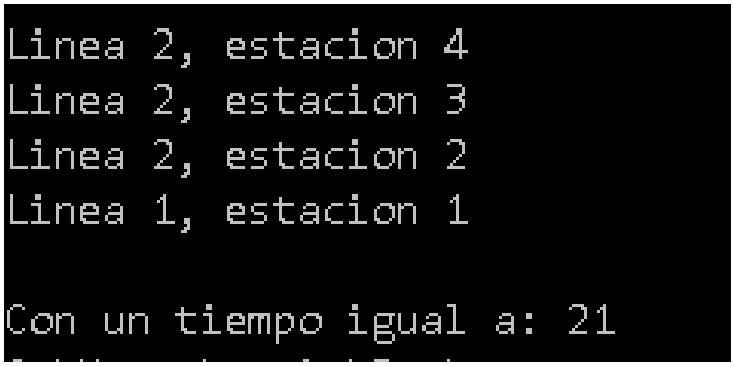


Figura 13: Resultado del ejemplo 1

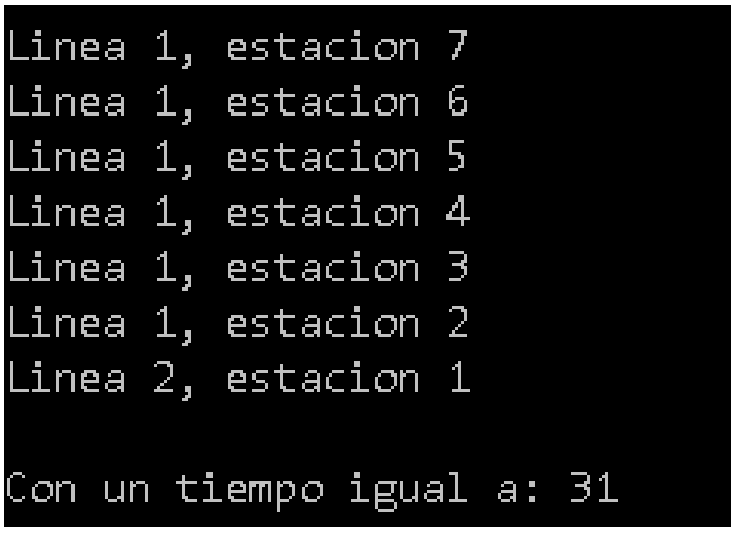


Figura 14: Resultado del ejemplo 2

4. Conclusiones

- Conclusiones generales:

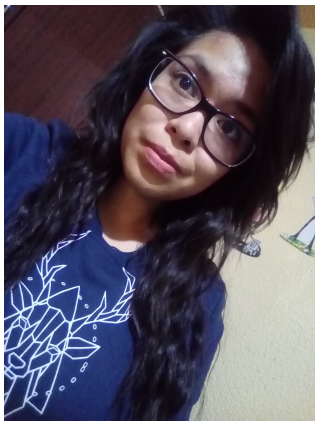
  En cuanto al programa de la subsecuencia común más larga de no aplicar la técnica de DP sería muy tardado, además este programa es considerado para textos no muy largos, ya que para asignar el tamaño de las tablas se consideran enteros, sin embargo esto podría ampliarse a

long long int para que soporte archivos mucho más grandes y la técnica y complejidad serían las mismas.

Para el programa de líneas de producción, el uso de las tablas permitió que fuera más sencillo obtener la ruta más corta ya que se sigue el rastro de la línea que se debe seguir con la tabla I, y gracias a la tabla f se puede obtener la suma total que da la ruta más corta.

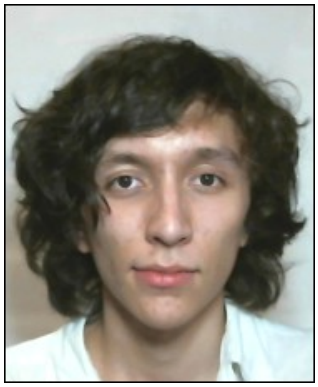
- Diana Paola De la Cruz Sierra

  La programación dinámica hace que la complejidad de los programas se reduzcan mucho, ya que generalmente hacen pasar de una complejidad cuadrática a una líneal, por lo que es un tema muy importante y de gran utilidad.



- Raya Chávez Samuel Antonio

  Con la programación dinámica, que va algo relacionado con los algoritmos Greedy pero no terminan siendo lo mismo, se pueden crear algoritmos con códigos más reducidos y por lo mismo mucho más eficientes, a comparación de los algoritmos que se crean en forma lineal y con varios ciclos para cumplir el mismo cometido.



5. Bibliografía

virtuniversidad. (S.F). Programación Dinámica. [PDF]. Disponible en:
https://www.virtuniversidad.com/greenstone/collect/informatica/archives/HASH0106.dir/doc.pdf

Facultad de estudios a distancia. (S.F). Capitulo 4: programación dinámica.[PDF]. Disponible en:
http://virtual.umng.edu.co/distancia/ecosistema/ovas/ingenieria-civil/investigacion-de-operaciones-ii/unidad-4/DM.pdf

Fochessato, N.(2015). Programación Dinámica. [PDF]. Disponible en: https://tc-arg.tk/pdfs/2015/02-DP.pdf

Gaston. (2016). Subsecuencia común más larga (LCS). diciembre 27, diciembre, de OIA Sitio web: https://oiaunlam.wordpress.com/2016/05/09/lcs/

Benoso, B. (2020). Líneas de producción mediante programación dinámica. diciembre 27, 2020, de Youtube Sitio web: https://www.youtube.com/watch?v=7CuO_bvc5Hs