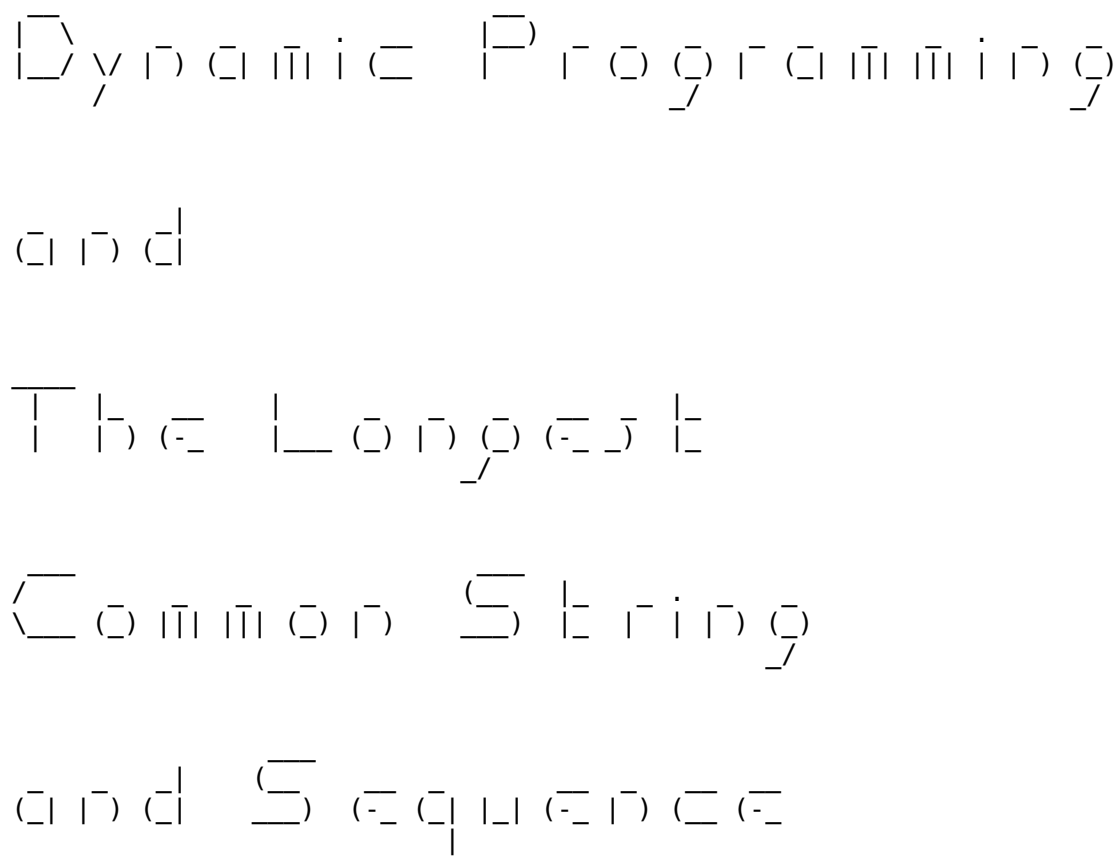
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬ 1. Introducción.

Ya hemos estudiado el uso de la programación dinámica para resolver varios problemas, incluyendo el knapsack. Veamos algunas de sus características.

La programación dinámica es útil cuando se está tratando de maximizar o minimizar un problema sujeto a una restricción.

En el problema del knapsack, dada una restricción en el peso de la mochila, se trata de encontrar la carga de más alta ganancia.

En el problema del itinerario de viaje, dada la restricción de los días disponibles, se trata de encontrar los lugares que se deben visitar, con el mayor nivel de rating.

Se puede utilizar programación dinámica cuando los problemas se pueden descomponer en subproblemas, y los subproblemas no dependen unos de otros. Es decir, son independientes.

▬▬▬ 2. Forma General de la Solución.

Puede ser difícil encontrar la fórmula general para un problema de programación dinámica. Pero existen unas consideraciones generales que se pueden tomar en cuenta.

Todo problema de DP utiliza un matriz de resultados.

Los valores en las celdas son usualmente lo que se trata de maximizar o minimizar.

Cada celda es un subproblema, así que se debe pensar como se puede dividir el problema en estas celdas.

Veamos ahora otro ejemplo, que utiliza una nueva fórmula para calcular las celdas.

▬▬▬ 3. Problema de Dictionary.com.

Ocasionalmente alguien escribe una palabra en algún buscador de internet, o en un diccionario en línea para encontrar una definición. Cuando esta palabra se escribe de forma incorrecta, el buscador sugiere una búsqueda con una palabra que se supone es la correcta.

En nuestro ejemplo vamos a suponer que estamos en un sitio que realiza traducciones y que al escribir una palabra incorrecta, se nos va a sugerir la palabra correcta.

Supongamos que escribimos la palabra “hish”.

Sin embargo, queríamos la traducción de la palabra “fish”.

Por error escribimos “hish”.

Cuál de estas dos palabras puede ser la correcta:
“hish” o “fish”.

Inicialmente, vamos a resolver el problema denominado “the longest common string”.

```
h i s h
  ▼ ▼ ▼
f i s h
```

Tienen un string de tamaño 3.
Esta debe ser la respuesta de nuestro problema.

▬▬▬ 4. The Longest Common String.

Para resolver el problema se construye una matriz con la palabra que se escribió en las columnas y la palabra contra la que deseamos hacer la comparación en las filas. El orden podría ser inverso, realmente no es importante. Lo importante es cómo se realizan los cálculos.

	h	i	s	h
f	--	--	--	--
i	--	--	--	--
s	--	--	--	--
h	--	--	--	--

Como en todos los casos de DP, los campos en la matriz es lo que deseamos optimizar. En este caso, entre mayor sea el número de letras en común, de manera continua, mayor será el número.

Encontrar un algoritmo en DP no es fácil. Pero tiene que ver con la fila anterior en cada etapa del problema. Veamos una descripción de la solución.

- Si las letras no son iguales, poner cero en la casilla.
- Si las letras son iguales, entonces
  la casilla de la fila anterior a la izquierda.

▬▬▬ 5. LCStr para “fish” y “hish”.

Veamos la construcción de la matriz para “fish” y “hish”.

	h	i	s	h

f	0	0	0	0
i	0	1	0	0
s	0	0	2	0
h	1	0	0	3

Observe que la solución está en la casilla final derecha y es de “3”.

▬▬▬ 6. LCStr para “fish” y “vista”.

Veamos que ocurre en este otro caso.

Veamos la solución en pseudocódigo:

```
if a[i] = b[j] then
  mat[i][j]: mat[i-1][j-1] + 1
else
  mat[i][j]: 0;
```

Supongamos que escribimos la palabra vista.
Aquí tenemos la matriz.

	v	i	s	t	a
f	0	0	0	0	0
i	0	1	0	0	0
s	0	0	2	0	0
h	0	0	0	0	0

Como se puede observar, la solución en este caso se encuentra en la fila “s”, columna “s”. Y tiene un valor de 2.

Para encontrar la solución se debe recorrer la pseudodiagonal y encontrar el máximo valor.

▬▬▬ 7. The Longest Common Sequence.

Supongamos que en el motor de búsqueda se escribió la palabra “fosh”.

Cuál de estas otras palabras es la mejor sugerencia, “fish” o “fort”?

Observemos que “fish” tiene 3 letras en común:

```
f o s h
▼   ▼ ▼
f i s h
```

Mientras que “fort” tiene únicamente 2 letras en común.

```
f o s h
▼ ▼
f o r t
```

Se tienen 3 letras en común. En este caso el algoritmo anterior, del “longest string” no funciona adecuadamente. Veamos las matrices.

>>> LCStr “fish” y “fosh”

	f	o	s	h
f	1	0	0	0
i	0	0	0	0
s	0	0	1	0
h	0	0	0	2

>>> LCStr “fort” y “fosh”

	f	o	s	h
f	1	0	0	0
o	0	2	0	0
r	0	0	0	0
t	0	0	0	0

▬▬▬ 8. LCSeq para “fosh”y “fish”.

En estos casos se utiliza el algoritmo “the longest sequence”, se utiliza denotarlo como LCSeq.

Veamos el pseudocódigo de este problema:

Si las letras son iguales entonces:

  tome valor de arriba-izquierda + 1
  (igual que en LCStr)

Si las letras son diferentes entonces:

  tome el máximo de la celda de arriba y
  la celda de la izquierda.

Otra forma de pseudocódigo:

```
if a[i] = b[j] then

   mat[i][j]: mat[i-1][j-1] + 1

else

   mat[i][j]: max( mat[i-1][j],
                   mat[i][j-1]
                 )
```

Veamos como queda la matriz en ambos casos.

>>> LCSeq para “fosh” y “fish”.

	f	o	s	h
f	1	1	1	1
i	1	1	1	1
s	1	1	2	2
h	1	1	2	3

LCSeq = 3
La respuesta para el “longest common sequence” es de 3.

>>> LCSeq “fosh”y “fort”.

	f	o	s	h
f	1	1	1	1
o	1	2	2	2
r	1	2	2	2
t	1	2	2	2

LCSeq = 2
La respuesta para el “longest common sequence” es de 2.

La recomendación para este problema sería la palabra “fish” y no la palabra de “fort”, pues la primera tiene más letras en común.

▬▬▬ 9. Análisis de Complejidad.

Para resolver estos problemas de forma exhaustiva, se requiere de revisiones que tienen un nivel de complejidad del orden de:

$O(2^n)$.

Al utilizar el algoritmo de programación dinámica, el tiempo de ejecución consiste en llenar la matriz. Si la primera palabra tiene “n” letras (las filas) y la segunda palabra tiene “m” letras, las columnas, entonces el tiempo de ejecución está dado por:

$O(n*m)$

▬▬▬ 10. Algunas Aplicaciones de LCStr y LCSeq.

Estos conceptos, que parecen muy sencillos, tienen una gran utilidad en la vida real. Por ejemplo.

• Los biólogos utilizan “longest common sequence” para encontrar similitudes en tiras a ADN. Se utilizan para saber que tan similares son dos animales o dos virus. Entre otras cosas se utiliza este algoritmo para encontrar nuevas curas, en especial para la enfermedad de esclerosis múltiple.

• Si alguna vez ha utiliza el comando “git diff”?. Diff indica las diferencias que existen entre dos archivos. Y utiliza programación dinámica.
https://git-scm.com/docs/git-diff

• En continuación al tema anterior, existe un algoritmo particular llamado “Levenshtein Distance”, que mide que tan similares son dos strings de gran tamaño.

Este algoritmo se utiliza para muchas cosas, para hacer “spell-checking”, hasta para saber si dos archivos, aunque hayan sido modificados, son un copia.

• Cuando se utiliza un procesador de texto, las líneas se acomodan de acuerdo a un algoritmo llamado “word wrap”. Para medir que las líneas de texto queden ordenadas de forma consistente se utiliza programación dinámica.

▬▬▬ 11. Resumen.

• El algoritmo de “longest common string” nos indica el máximo largo de un string dentro de otro string, sin separaciones.

• El algoritmo de “longest common sequence” nos indica el máximo largo de un string dentro de otro string, con separaciones.

• Ambos algoritmos tienen una gran cantidad de aplicaciones en la vida real.

▬▬▬ 12. Ejercicios.

●● Ejercicio 1.

Haga la tabla de LCStr para las palabras “blue” y “clue”.

●● Ejercicio 2.

Haga la tabla de LCStr para las palabras “blue” y “clues”.

●● Ejercicio 3.

Haga la tabla de LCSeq para las palabras “blue” y “clue”.

●● Ejercicio 4.

Haga la tabla de LCSeq para las palabras “blue” y “clues”.

●● Ejercicio 5.

Implemente el algoritmo de LCStr en un lenguaje de programación.

●● Ejercicio 6.

Implemente el algoritmo de LCSeq en un lenguaje de programación.