ANÁLISIS

Se dispone de dos secuencias de caracteres S[1..n] y T[1..m]. Se debe conocer cuál es la subsecuencia común máxima entre ambas; es decir, el número de caracteres máximo que aparecen en el mismo orden (no necesariamente juntos) en ambas secuencias.

DISEÑO DEL ALGORITMO

- **El problema propuesto es un problema de optimización**, dado que se debe maximizar el número de caracteres consecutivos en común en dos subsecuencias.
- **Se puede resolver por etapas**: En cada etapa comprobaremos una subsecuencia S[1..i] con otra T[1..j].
- **Ecuación recurrente**:

    - **Variables:**
        - i: Caracteres desde el 1 hasta el i de la secuencia S que se están considerando para comparar con la otra subsecuencia.
        - j: Caracteres desde el 1 hasta el j de la secuencia T que se están considerando para comparar con la otra subsecuencia.
        - LCS(i,j): El número de caracteres de la subsecuencia máxima en común desde S[1..i] y T[1..j].
    - **Valor objetivo:**
        - Se desea conocer LCS(n,m)
    - **Caso/s base:**
        - El caso base se dará cuando alguna secuencia (S o T) sea vacío. En este caso, LCS(0,0)= 0, LCS(i,0)= 0, LCS(0,j)= 0
    - **Caso general:**
        - En LCS(i,j) hemos supuesto que ya están comparados todas las subsecuencias de S[0..i-1] y T[0..j-1]. Puede ocurrir que S[i]=T[j], por lo que en este caso hay que añadir 1 al tamaño de la subsecuencia común:
            - LCS(i,j)= LCS(i-1, j-1)+1
        - En caso de que S[i] <> T[j], tendríamos que verificar dos casos:
            - LCS(i,j)= LCS(i-1, j) → La subsecuencia más larga puede ser aquella que compara S[0...i-1] con T[0...j].
            - LCS(i,j)= LCS(i, j-1) → La subsecuencia más larga puede ser aquella que compara S[0...i] con T[0...j-1].

        De este modo, obtenemos la ecuación:

$$LCS(i,j)=\begin{pmatrix} LCS(i-1,j-1)+1\,;si\,S_i=T_j \\ max\{LCS(i-1,j),LCS(i,j-1)\,;si\,S_i\neq T_j\} \end{pmatrix}$$

- **Verificación del cumplimiento del P.O.B:** Para el caso base en el que alguna de las secuencias es vacía, la máxima subsecuencia común es 0, y el valor es óptimo. Para el caso general el algoritmo devuelve el valor LCS(i,j), que se asume óptimo. Si no lo fuera:
    - Cuando S_i <> T_j tenemos que la máxima subsecuencia se obtiene como el máximo entre LCS(i-1,j) y LCS(i, j-1) que deberían no ser óptimos. Esto no es posible, dado que el

algoritmo siempre ha ido escogiendo el máximo entre ambos valores para construir la solución.
  - Cuando $S_i = T_j$, la máxima subsecuencia es de al menos valor 1 (por el carácter i de S y el carácter j de T), a los que hay que sumar la subsecuencia máxima de LCS(i-1, j-1). Este valor siempre se ha construido escogiendo el máximo entre LCS(i-1,j) y LCS(i,j-1) cuando los caracteres previos de S y de T no coincidían, y sumando +1 cuando sí lo eran. Por tanto, no podría existir una subsecuencia mayor que la dada por la ecuación en recurrencias.
  - Atendiendo al razonamiento anterior, se cumple el Principio de Optimalidad del Bellman, que indica que si una secuencia de decisiones es óptima, entonces todas sus subsecuencias de decisiones también lo son.

- **Diseño de la memoria:**

  - Para resolver el problema, LCS(i,j) se representará como una matriz.
  - La matriz tendrá n+1 filas, indexadas de 0..n. Cada fila i estará asociada a un carácter de la cadena S (salvo el índice 0 que se asocia a cuando S fuese cadena vacía).
  - La matriz tendrá m+1 columnas, indexadas de 0..m. Cada columna j estará asociada a un carácter de la cadena T (salvo el índice 0 que se asocia a cuando S fuese cadena vacía).
  - Cada celda de la matriz LCS(i,j) contendrá el máximo número de caracteres en secuencia común entre S y T, desde el primer carácter 0 hasta el i/j, respectivamente para cada cadena.
  - La memoria se rellenará de la siguiente forma:
    - En primer lugar, se rellenan las celdas correspondientes a los casos base LCS(i,0)=LCS(0,j)= 0.
    - En segundo lugar, se rellenarán las filas {1, 2, …., n}, en orden secuencial.
    - Cada fila se rellenará en orden creciente de columnas {1...m}.

Con este diseño, construimos el algoritmo de Programación Dinámica como sigue:

```
ALGORITMO LCS, V= Subsecuencia( S[1...n], T[1..m] )

    LCS ← matriz de n+1 filas indexadas {0...n} y m+1 columnas indexadas {0..m}
    Para cada fila i en {0...n}, hacer:
        T(i,0)= 0
    Para cada columna j en {0...m}, hacer:
        T(0, j)= 0

    Para cada fila i en {1..n}, hacer:
        Para cada columna j en {1..m}, hacer:
            Si Si = Tj, hacer:
                LCS(i,j)= LCS(i-1, j-1) + 1
            En otro caso, hacer:
                LCS(i,j)= max{LCS(i-1,j), LCS(i,j-1)}

    V ← LCS(n, m)
    Devolver LCS, V
```

- **Recuperación de la solución:** La solución se recuperará desde el valor objetivo de la matriz calculada previamente, recorriendo hacia atrás la recurrencia. El siguiente algoritmo devuelve la solución Sol (el conjunto de caracteres en común):

```
ALGORITMO Sol= RecuperaSubsecuencia( LCS[0..n, 0..m], S, T )

    Sol ← ∅
    i ← n, j ← m
    Mientras i <> 0  y j<>0, hacer:
        Si S[i] = T[j], hacer:
            Insertar S[i] al principio de Sol
            i ← i-1 , j ← j-1
        En otro caso, hacer:
            Si LCS(i,j)= LCS(i-1,j), hacer:
                i ← i-1
            en otro caso, hacer:
                j ← j-1
    Devolver Sol
```

DETALLES DE IMPLEMENTACIÓN

En la solución planteada en el código fuente hay un único fichero **main.cpp** conteniendo:

- **Función PD, que implementa secuencialmente los algoritmos *subsecuencia* y *recuperasubsecuencia* previos.**

El funcionamiento del programa es el siguiente:

1. Se leen dos cadenas de caracteres S y T desde entrada
2. Se llama al algoritmo PD, que devuelve el coste y la recuperación de la solución.
3. Se muestran los resultados.

Adicionalmente, se ha construido un fichero makefile para construir el programa de forma simple. Se compila y ejecuta de la siguiente forma:

1. Nos situamos en la carpeta donde está el código fuente y, desde la terminal, escribimos: ***make***
2. Ejecutamos el programa desde la terminal escribiendo: ***./Solucion***
3. Se nos pedirá por consola que introduzcamos dos cadenas. Tras introducirlas, se nos mostrará por pantalla la solución al problema.