REPORTE DIVIDE Y VENCERAS

LONGEST COMMON SUBSEQUENCE

Centro Universitario de Ciencias Exactas e Ingenierías
Departamento: División de Tecnologías para la Integración Ciber-Humana
Materia: Análisis de algoritmos
Profesor: Jorge Ernesto López Arce Delgado
Alumnos: De la Mora Villaseñor Diego Gabriel
Lopez Esparza Angel Emanuel
López Galván Melanie Montserrat
Código: 220576197
Carrera: Ingeniería en Computación
Sección: D05
Fecha: 26/10/2025

Contenido

Introducción

Divide y vencerás se trata de una estrategia de diseño de algoritmos, en la cual se busca dividir el problema en principal en subproblemas más sencillos, cada uno de los problemas se resuelve por separado, hasta que se terminan de resolver todos los subproblemas; cuando todos los subproblemas fueron resueltos, las soluciones se unen de una manera inteligente para obtener la resolución del problema general.

Podemos representar la estrategia en 3 etapas:

Dividir: El proceso en el que el problema principal se divide en nuevos subproblemas.

Vencer: El proceso en que cada uno de los subproblemas se resuelve por separado.

Combinar: El proceso en que los resultados de los subproblemas se combinan la para solución del problema original.

La relación que existe entre la programación dinámica y divide y vencerás, es que hay veces que los subproblemas que se generan en el acto de dividir, pueden presentarse en otras de las ramas de otra división. Cuando esto ocurre, entonces a la hora de resolver los subproblemas, se estaría resolviendo el mismo problema varias veces en un mismo intento de obtener la solución general del problema.

La programación dinámica busca solucionar el problema utilizando un espacio en memoria dinámico, en donde se almacenarán los resultados de cada uno de los subproblemas, con la finalidad que que si aparece el mismo subproblema otra vez, se puede acceder directamente a la solución en lugar de calcularlo nuevamente. Ahorrando tiempo en la ejecución, pero a costa de gastar más memoria. Para mostrar un ejemplo de ambas estrategias de diseño, se pone a prueba el problema de LCS.

El problema de LCS surge alrededor de los años 70’s en donde los investigadores en el campo de la bioinformática se presentaron con un problema con respecto a las cadenas de ADN presentes en los organismos vivos. Era necesario la creación de un algoritmo capaz de comparar 2 secuencias de entrada y obtener que tan parecido eran entre ellas.

El algoritmo de fuerza bruta/divide y vencerás que se diseñó primero, consistía en la generación de todas las subsecuencias posibles en cada una de las 2 cadenas. Cada una de las cadenas puede tener 2^n combinaciones, por lo que al final, este algoritmo presentaba una complejidad de O(2^n*2^m), donde n es el tamaño de la cadena A y m el tamaño de la cadena B. Con una estrategia de divide y vencerás sin ningún tipo de mejora, requiere la generación de cada subsecuencia posible por cada una de las cadenas (Etapa de división), comparar cada una de las subsecuencias por todas y cada una de las subsecuencias de la cadena B (Etapa de conquista), y comparar las subsecuencias mas largas para quedarse con la mayor (Etapa de combinar).

El algoritmo de Needleman-Wunsch, publicado por Saul Needleman y Christian Wunsch en 1970, fue uno de los primeros intentos de aplicación de programación dinámica para la resolución del problema LCS, el algoritmo consiste en la generación de una matriz de N+1*M+1, en la que se colocaran las puntuaciones (Que tan larga es una subsecuencia) de todas las subsecuencias generadas. El algoritmo compara un carácter i de la cadena A con un carácter j de la cadena B. Si encuentra que ambos caracteres son iguales se copia el valor de la puntuación de i-1 e j-1 y se le suma 1 para llenar el espacio ij de la matriz generada, si no se logra encontrar una conciencia, entonces se toma la puntuación más grande de la anterior comparación entre las 2 cadenas, es decir i-1,j o i,j-1. Toda la matriz es recorrida, lo que equivale a que cada carácter de ambas cadenas fue comparado con cada carácter de las otra. Al terminar el algoritmo, se garantiza que la puntuación de coincidencia, es decir, el tamaño de la subsecuencia más larga que puede haber, se encuentra en la posición n,m.

Objetivos

- Comprender el uso de técnicas de programación como lo son ‘Divide y Vencerás’ y ‘Programación Dinámica’
- Aplicar el algoritmo de ‘longest common subsequence’
- Ver en que se emplea el algoritmo en la vida real
- Comparar los resultados entre las dos formas de implementación del algoritmo

Desarrollo

El planteamiento del algoritmo parte de una premisa muy sencilla, la cual es el núcleo de todo el algoritmo, las subsecuencias de elementos. Estas forman parten de una secuencia original, donde se obtienen eliminando algún, o ninguno, de los elementos de la secuencia original; pero sin modificar el orden de estos elementos. No necesitan un orden consecutivo.

Tenemos un arreglo principal ‘A, B, C, D, E’ y a partir de este arreglo se pueden crear subsecuencias, por ejemplo ‘A, C, E’. Es una subsecuencia valida del arreglo principal, ya que los elementos mantienen el mismo orden, la A va primero, luego la C y por último la E. En caso contrario; si por ejemplo creáramos la subsecuencia ‘A, E, C’, entonces ya no seria válida, ya que la letra E se encuentra antes que la C y eso no es real en la cadena principal.

Lo que define si una subsecuencia es válida es cuando ninguno de los elementos antecede al orden de otro elemento en la cadena principal, sin importar si se eliminó cualquier otro elemento en medio de ellos.

Pseudocódigo implementación Divide y Vencerás:

```
FUNCIÓN lcs_recursivo(X, Y, m, n)
    SI m == 0 O n == 0 ENTONCES
        RETORNAR 0
    FIN SI
    SI X[m-1] == Y[n-1] ENTONCES
        RETORNAR 1 + lcs_recursivo(X, Y, m-1, n-1)
    SINO
        RETORNAR max( lcs_recursivo(X, Y, m, n-1), lcs_recursivo(X, Y, m-1, n) )
    FIN SI
FIN FUNCIÓN
```

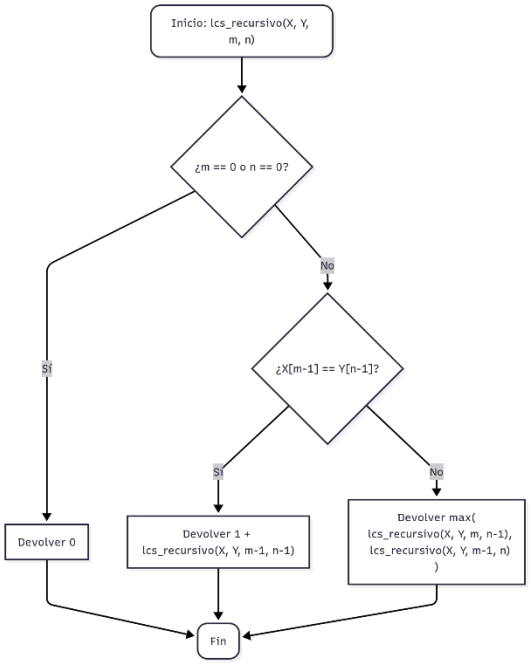

Fig. 1. Diagrama implementación Divide y Vencerás:

Pseudocódigo implementación programación dinámica:

```
FUNCIÓN lcs_dinamico_tabla(X, Y)
    m = longitud(X)
    n = longitud(Y)
    CREAR L[0..m][0..n]
    INICIALIZAR L con 0s
    PARA i DESDE 1 HASTA m
        PARA j DESDE 1 HASTA n
            SI X[i-1] == Y[j-1] ENTONCES
                L[i][j] = L[i-1][j-1] + 1
            SINO
                L[i][j] = max(L[i-1][j], L[i][j-1])
            FIN SI
        FIN PARA
    FIN PARA
        RETORNAR (L, L[m][n])
FIN FUNCIÓN
```

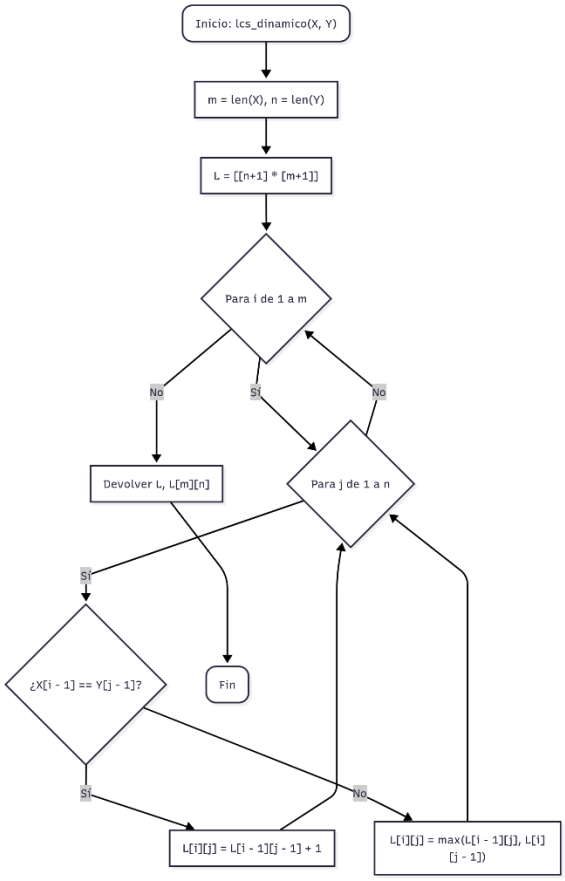

Fig. 2. Diagrama implementación programación dinámica.

En cuanto a complejidad algorítmica, este es un problema NP-Completo, y en fuerzas bruta su complejidad algorítmica seria de $O(2^{n+m})$, donde realiza todas las iteraciones del algoritmo. En la implementación de divide y vencerás este solo toma en cuenta el mínimo de N u M, donde su complejidad sería $O(2^{min(n,m)})$, esto ocurre porque calcula ramas innecesarias que se podrían evitar calcular. Esto lo arregla de mejor manera la programación dinámica, donde ahora al cortar ramas que ya se habían calculado la complejidad algorítmica mejora mucho más, está siendo O(n*m).

En los resultados se observa que efectivamente la programación dinámica es mucho más rápida que la propia implementación con divide y vencerás, donde en la gráfica se ve una mejoría del 99%, con el coste de aumentar la memoria empleada.

Fig. 3. Ejecución y graficas del programa.

Con muchos más datos la diferencia es realmente grande, donde su forma dinámica apenas tarda en realizar el cálculo a comparación de la implementación en divide y vencerás, aunque efectivamente emplea más memoria, como se muestra en la figura 4.

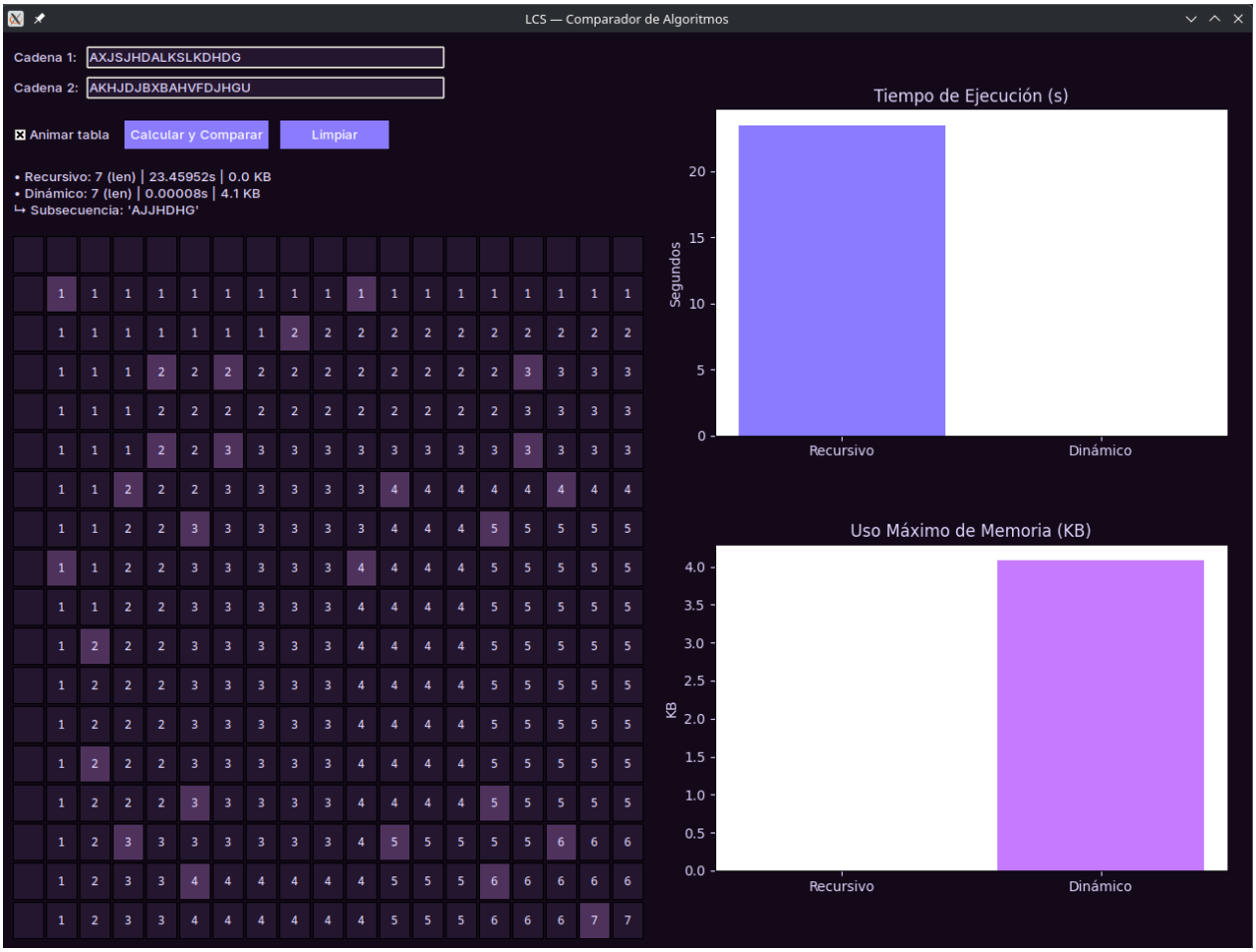

Fig. 4. Ejecución con cadenas más grandes.

Conclusiones

Estas técnicas no se pueden aplicar a todos los tipos de problemas, algo que aprendimos después de cambiar el algoritmo y ver que la implementación de ‘Divide y Vencerás’ y la implementación de ‘Programación Dinámica’ no encajaban muy bien en el problema, quizás podríamos haber agregado en algunas partes del algoritmo algo de divide y vencerás. Por lo que al cambiar de algoritmo vimos que existía el que programamos, la subsecuencia común más larga, el cual tiene aplicaciones realmente interesantes, donde se usa en sistemas de control de versiones o en bioinformática comparando subsecuencias de genomas, poniendo algunos ejemplos. Se nos hizo interesante ver el gran cambio que conlleva implementar programación dinámica y actualmente la memoria no es un gran problema por el cual preocuparse.

Conclusión Emanuel: La estrategia de diseño de divide y vencerás para algoritmos demuestra ser una extensión natural de la manera en que resolvemos problemas en la vida diaria. Aunque no siempre pueden ser las más eficientes, su interpretación nos permite una mejor comprensión de los problemas. Además, aprendí nuevas formas en las que puede representarse el uso de la programación dinámica, no siendo siempre una lista para almacenar soluciones, si no que se pueden tener representaciones matriciales.

Conclusión Diego: Me enoje por cambiar el algoritmo, pero era necesario, lo juro… El nuevo algoritmo, el que realizamos para la práctica, es bastante interesante y si tiene aplicaciones reales. No era complicado implementarlo y me agrada mucho que la implementación de programación dinámica mejora mucho el tiempo.

Conclusión Melanie: Aunque este algoritmo pueda ser sencillo para implementar tiene un peso matemático detrás que permitió que fuera implementado o probado hace varios años y sigue manteniendo una relevancia, especialmente en el ámbito de la biología que fue nuestro enfoque principal

Referencias

Wikipedia. (2025, October 26). *Longest common subsequence*. In *Wikipedia, The Free Encyclopedia*. Retrieved from https://en.wikipedia.org/wiki/Longest_common_subsequence