Proyecto II: Reporte Escrito

Ricardo Víquez Mora Escuela de Computación
Tecnológico de Costa Rica
Sede Interuniversitaria de Alajuela
Costa Rica
Email: rychield@gmail.com

TABLA DE CONTENIDOS

Proyecto II: Reporte Escrito

***Resumen*—En este reporte, se comparó el desempeño en tiempos de corrida para algoritmos que solucionan el problema de la mochila y también para los que resuelven el problema del ordenamiento de secuencias. Para cada problema, hay 2 versiones del algoritmo que lo resuelven (algoritmo de fuerza bruta y algoritmo de programación dinámica); siendo un total de 4 algoritmos a analizar. Por medio del análisis de código de los algoritmos, y de los gráficos del desempeño de estos, se obtuvo evidencias para explicar la complejidad temporal de cada algoritmo respectivo.**

I. Introducción

Como parte de un proyecto del curso de investigación de operaciones, se programaron 4 algoritmos: algoritmo de la mochila (fuerza bruta), algoritmo de la mochila (programación dinámica), algoritmo del ordenamiento de secuencias (fuerza bruta) y algoritmo del ordenamiento de secuencias (programación dinámica). El objetivo de este reporte es determinar la complejidad temporal de cada algoritmo, por medio del análisis de código y de los tiempos de corrida de los algoritmos.

A. Problema de la mochila

Se trata de un problema de optimización combinatoria, lo que quiere decir que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema [1]. Se tienen n elementos distintos, y un contenedor que soporta una cantidad especifica de peso W. Cada elemento i tiene un peso w i , un valor o beneficio asociado dado por b i , y una cantidad dada por c i.

El problema consiste en agregar elementos al contenedor de forma que se maximice el beneficio de los elementos que contiene sin superar el peso máximo que soporta. La solución debe ser dada con la lista de los elementos que se ingresaron y el valor del beneficio máximo obtenido.

Ejemplo: Con un contenedor de peso W = 50. 3 artículos con pesos: 5, 15, 10, beneficios: 20, 50, 60, y unidades 4, 3 y 3 respectivamente. Tiene como solución un beneficio máximo de 260 agregando los artículos 3 (3 unidades) y 1 (4 unidades).

B. Problema del ordenamiento de secuencias

Consiste en alinear 2 secuencias compuestas de las letras correspondientes a los ácidos nucleicos (A,T,G y C), de tal forma que la similitud entre ambas secuencias sea la mejor. Para esto se pueden introducir gaps en alguna de las secuencias (en caso de que sean de distinto tamaño), y/o posicionar alguna subsecuencia de tal forma que calce con la mayor cantidad de bases de la otra secuencia. Siempre se usará el método de scoring +1, -1, -2 para cada par de posiciones en las 2 secuencias (posiciones iguales, distintas, o con gap, respectivamente).

Para la solución hecha con programación dinámica, tenemos que el algoritmo de Needleman-Wunsch garantiza la obtención del mejor alineamiento, además, "se suele utilizar en el ámbito de la bioinformática para alinear secuencias de proteínas o de ácidos nucleicos" [2].

II. Metodología

Para evaluar el desempeño de los algoritmos, se escogió el tiempo de corrida en segundos como unidad de medida del desempeño. Para tal fin, se utilizó la librería de python "time".

Para calcular los tiempos de corrida, cada algoritmo cuenta con experimentos de distintos parámetros, y en cada experimento se toma la mediana del tiempo de corrida a partir de 5 repeticiones del experimento respectivo. Asimismo, se analizó el código fuente de cada algoritmo, para determinar la complejidad temporal teórica respectiva. A continuación, se detalla el número de experimentos y los parámetros de estos, para el problema de la mochila y el de ordenamiento de secuencias.

A. Experimentos para los algoritmos de la Mochila

El algoritmo de fuerza bruta y el de programación dinámica, cuentan con 20 experimentos cada uno, los cuales utilizan los siguientes parámetros:

- Una mochila con un "peso máximo" de 100 uds.
- Un único tipo de artículo de "peso" 5 uds., "beneficio" 20 uds. y "cantidad" (1 a 20 uds.)

Donde cada experimento va incrementado el parámetro "cantidad" en 1 unidad. Por ejemplo, en el experimento #1 se asigna 1 unidad al parámetro "cantidad", y en el experimento #2 se asignan 2 unidades, sin cambios en los demás parámetros.

A continuación, se muestra el contenido del archivo de entrada para el experimento #1 y #20

Experimento #1:
100 (peso máximo)
5,20,1 (peso, beneficio y cantidad)

Experimento #20:
100 (peso máximo)
5,20,20 (peso, beneficio y cantidad)

B. Experimentos para los algoritmos del Ordenamiento de secuencias

El algoritmo de fuerza bruta y el de programación dinámica, cuentan con 11 experimentos cada uno, los cuales utilizan los siguientes parámetros:

- Secuencia1 compuesta de sólo letras "A" (con un largo de 1 a 11 caracteres).
- Secuencia2 compuesta de sólo letras "A" (con un largo de 1 carácter).

Donde cada experimento va incrementado el largo de la Secuencia1 en 1 carácter hasta alcanzar un largo de 11 caracteres (manteniendo con un largo de 1 carácter la Secuencia2). De modo que el experimento #1 tiene una Secuencia1 de largo 1 carácter y una Secuencia2 de largo 1; mientras que el experimento #11 tiene una Secuencia1 de largo 11 y una Secuencia2 de largo 1.

A continuación, se muestra el contenido del archivo de entrada para el experimento #1 y #11

Experimento #1:
A (Secuencia1)
A (Secuencia2)

Experimento #11:
AAAAAAAAAAA (Secuencia1)
A (Secuencia2)

III. Análisis de resultados

Fig. 1. Análisis de código del algoritmo de la mochila (fuerza bruta). El análisis de código indica una complejidad temporal de: $2^n + n^2 + n(O(n\ escoge\ p) + nlogn) + n = O(2^n)$.

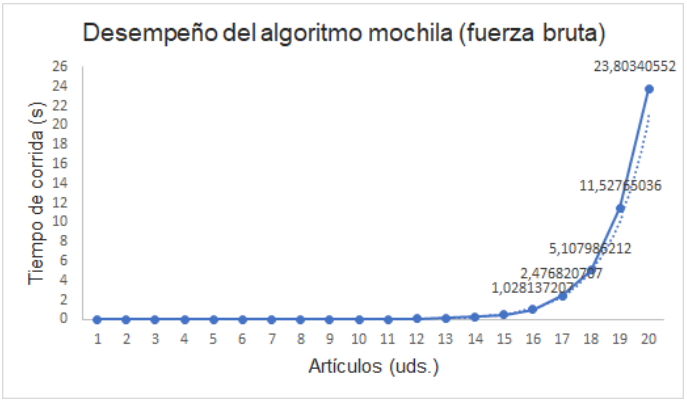

Fig. 2. Tiempos de corrida obtenidos con el algoritmo de la mochila de fuerza bruta. Para todos los experimentos, el peso máximo de la mochila se mantiene constante, pero la cantidad de artículos de entrada aumenta. Además, la cantidad de artículos coincide con el ID del experimento respectivo.

En la Figura 1, se determinó una complejidad temporal de $O(2^n)$ con base en el análisis del código del algoritmo de la mochila (fuerza bruta). Esta misma complejidad temporal se manifiesta en la Figura 2, donde los tiempos de corrida muestran un incremento exponencial con respecto a la cantidad de artículos de entrada en el problema a resolver. Por lo tanto, no se rechaza la hipótesis de que la complejidad temporal del algoritmo es $O(2^n)$.

Fig. 3. Análisis de código del algoritmo de la mochila (programación dinámica). El análisis de código indica una complejidad temporal de: $n + m + n * m + n * m + n + nlogn = O(n * m)$. Donde "n" es función de la cantidad de artículos de entrada, y "m" es función del peso máximo de la mochila.

En la Figura 3, se determinó una complejidad temporal de $O(n * m)$ con base en el análisis del código del algoritmo de la mochila (programación dinámica). Sin embargo, esta complejidad temporal no se observa en la Figura 4, donde los tiempos de corrida muestran un incremento lineal y no cuadrático, con respecto a la cantidad de artículos de entrada en el problema a resolver. Lo anterior se puede explicar, pues la compleji-

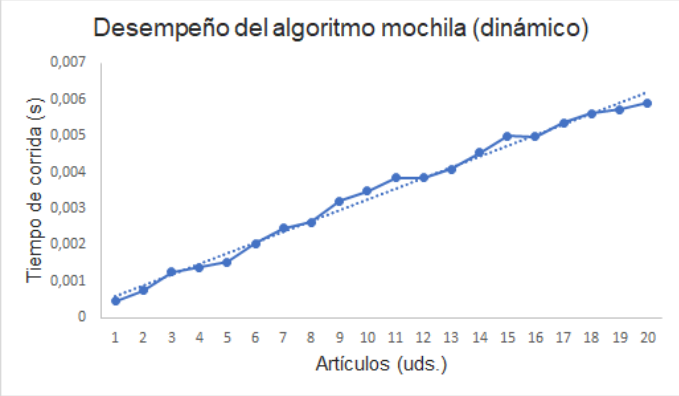

Fig. 4. Tiempos de corrida obtenidos con el algoritmo de la mochila de programación dinámica. Para todos los experimentos, el peso máximo de la mochila se mantiene constante, pero la cantidad de artículos de entrada aumenta. Además, la cantidad de artículos coincide con el ID del experimento respectivo.

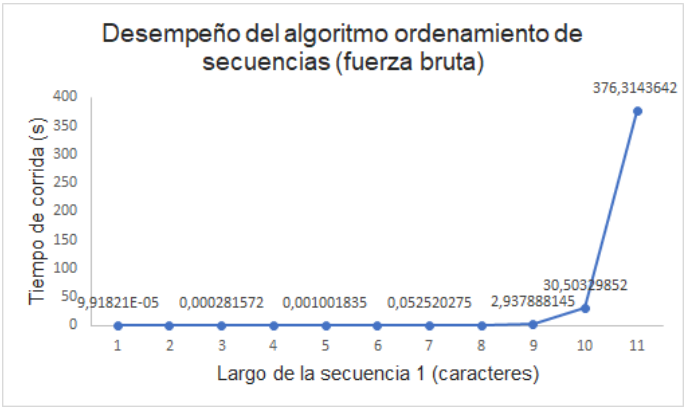

Fig. 6. Tiempos de corrida obtenidos con el algoritmo del ordenamiento de secuencias de fuerza bruta. Para todos los experimentos, el largo de la secuencia 1 aumenta, pero el largo de la secuencia 2 se mantiene constante. Además, el largo de la secuencia 1 coincide con el ID del experimento respectivo.

dad temporal de éste algoritmo dinámico depende tanto de la cantidad de artículos "n" como del peso máximo de la mochila "m"; pero en este caso, como todos los experimentos realizados en la Figura 4 mantienen un peso máximo constante, la complejidad temporal depende únicamente de la cantidad de artículos de entrada "n", por lo que se presenta un comportamiento lineal en vez de cuadrático. Por lo tanto, no se rechaza la hipótesis de que la complejidad temporal del algoritmo es $O(n * m)$.

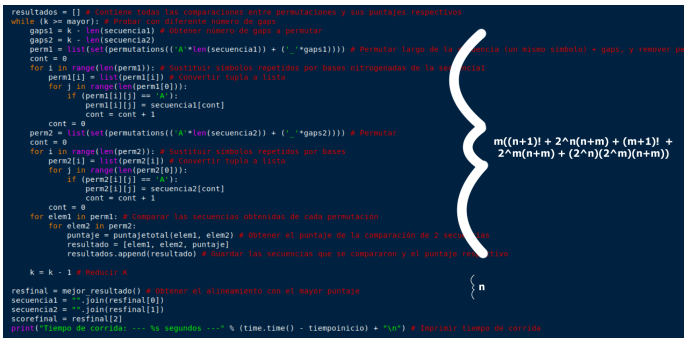

Fig. 5. Análisis de código del algoritmo de ordenamiento de secuencias (fuerza bruta). El análisis de código indica una complejidad temporal de: $m((n+1)! + 2^n(n+m) + (m+1)! + 2^m(n+m) + 2^n 2^m(n+m) + n) = O(m(n+1)!)$, asumiendo que $m <= n$. Donde "n" es función del largo en caracteres de la secuencia 1, y "m" es función del largo en caracteres de la secuencia 2.

En la Figura 5, se encontró una complejidad temporal de $O(m(n+1)!)$ (asumiendo que $m <= n$) con base en el análisis del código del algoritmo de ordenamiento de secuencias (fuerza bruta). Si bien es cierto que en la Figura 6, los tiempos de corrida muestran claramente un incremento factorial cuando el largo de la secuencia 1 es de 9, 10 y 11 caracteres respectivamente; sólo se muestra un comportamiento factorial y no $m(n+1)!$. Esto se debe a que en este caso específico, para todos los experimentos de la Figura 6, el largo de la secuencia 2 que corresponde a "m" se mantiene constante con un largo de 1 carácter. Por lo que para este caso, el comportamiento presentado en la Figura 6 corresponde a $(n+1)!$ y no a $m(n+1)!$. Entonces, no se rechaza la hipótesis de que la complejidad temporal del algoritmo es $O(m(n+1)!$, asumiendo que $m <= n$.

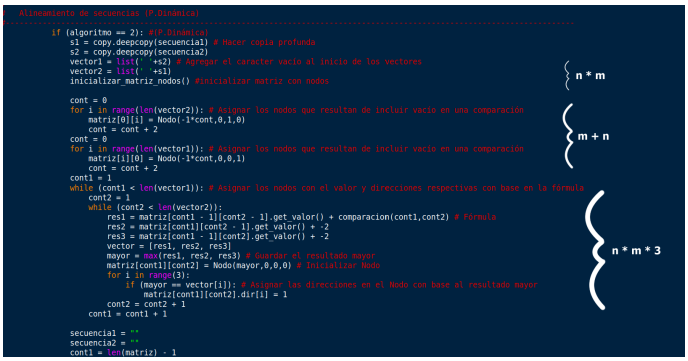

Fig. 7. Análisis de código del algoritmo de ordenamiento de secuencias (programación dinámica); se muestra la primera parte del código. El análisis completo indica una complejidad temporal de: $n*m+(m+n)+n*m*3+n*m+(n+n) = O(n*m)$. Donde "n" es función del largo en caracteres de la secuencia 1, y "m" es función del largo en caracteres de la secuencia 2.

En la Figura 7 y Figura 8, se determinó una complejidad temporal de $O(n * m)$ con base en el análisis del código del algoritmo de ordenamiento de secuencias (programación dinámica). Sin embargo, esta complejidad temporal no se manifiesta en la Figura 9, donde los tiempos de corrida muestran un incremento lineal y no cuadrático, con respecto al largo de la secuencia 1. Lo anterior se debe a que la complejidad temporal de éste

```
secuencia1 = ""
secuencia2 = ""
cont1 = len(matriz) - 1
cont2 = len(matriz[0]) - 1
while (cont1 > 0 or cont2 > 0): # Recorrer la ruta de la matriz (del final al inicio)
    if (matriz[cont1][cont2].get_diagonal()==1): # Se escoge el camino de la diagonal
        secuencia1 = secuencia1 + vector2[cont2]
        secuencia2 = secuencia2 + vector1[cont1]
        cont1 = cont1 - 1
        cont2 = cont2 - 1
    elif (matriz[cont1][cont2].get_izquierda()==1): # Se escoge el camino de la izquierda
        secuencia1 = secuencia1 + vector2[cont2]
        secuencia2 = secuencia2 + "_"
        cont2 = cont2 - 1
    elif (matriz[cont1][cont2].get_arriba()==1): # Se escoge el camino de arriba
        secuencia1 = secuencia1 + "_"
        secuencia2 = secuencia2 + vector1[cont1]
        cont1 = cont1 - 1

secuencia1 = string_reverso(secuencia1) # Obtener el reverso de las secuencias
secuencia2 = string_reverso(secuencia2)
scorefinal = matriz[len(matriz)-1][len(matriz[0])-1].get_valor() # Obtener score final del alineamiento
print("Tiempo de corrida: --- %s segundos ---" % (time.time() - tiempoinicio) + "\n") # Imprimir tiempo de corrida

imprimir_salida_alineamiento2() # Guardar en archivo de salida
```

n * m

n + n

Fig. 8. Análisis de código del algoritmo de ordenamiento de secuencias (programación dinámica); se muestra la segunda parte del código. El análisis completo indica una complejidad temporal de: $n*m+(m+n)+n*m*3+n*m+(n+n) = O(n*m)$. Donde "n" es función del largo en caracteres de la secuencia 1, y "m" es función del largo en caracteres de la secuencia 2.

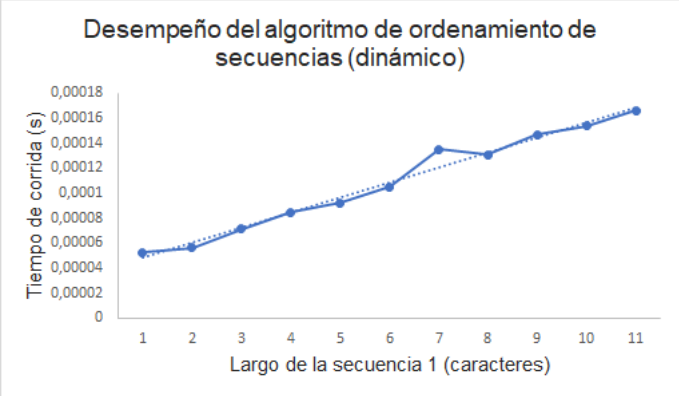

Fig. 9. Tiempos de corrida obtenidos con el algoritmo del ordenamiento de secuencias de programación dinámica (Needleman-Wunsch). Para todos los experimentos, el largo de la secuencia 1 aumenta, pero el largo de la secuencia 2 se mantiene constante. Además, el largo de la secuencia 1 coincide con el ID del experimento respectivo.

algoritmo dinámico depende tanto del largo de la secuencia 1 "n" como del largo de la secuencia 2 "m"; pero en este caso, como todos los experimentos realizados en la Figura 9 mantienen el largo de la secuencia 2 constante, la complejidad temporal depende únicamente del largo de la secuencia 1 "n", por lo que se presenta un comportamiento lineal en vez de cuadrático. Por lo tanto, no se rechaza la hipótesis de que la complejidad temporal del algoritmo es $O(n*m)$.

IV. Conclusión

Al analizar los gráficos de los tiempos de corrida, se encontraron evidencias que dan soporte práctico a las complejidades temporales estimadas con los análisis de código realizados a cada algoritmo implementado. Se logró determinar que la complejidad temporal del algoritmo de la mochila (fuerza bruta) tiene que ser exponencial $O(2^n)$, la del algoritmo de ordenamiento de secuencias (fuerza bruta) es del orden factorial $O(m(n+1)!$ (asumiendo que $m <= n$); y que los dos algoritmos de programación dinámica abarcados, presentan una complejidad temporal de $O(n*m)$.

Bibliografía

[1] "Problema de la mochila", Es.wikipedia.org, 2020. [Online]. Disponible: https://es.wikipedia.org/wiki/Problema_de_la_mochila. [Accedido: 11- Dec- 2020].

[2] "Algoritmo Needleman-Wunsch", Es.wikipedia.org, 2020. [Online]. Disponible: https://es.wikipedia.org/wiki/Algoritmo_Needleman-Wunsch. [Accedido: 13- Dec- 2020].