Desempeño de la Programación Dinámica

Segundo Proyecto de Investigación de Operaciones

Walter Morales
Escuela de Ingeniería en Computación
Unidad Desconcentrada de Alajuela
Carrera de Ingeniería en Computación
Alajuela, Costa Rica

I. Introducción

El presente documento corresponde al desarrollo y análisis de resultados del segundo proyecto del curso de Investigación de Operaciones IC-6400, brindado en el Instituto Tecnológico de Costa Rica durante el segundo semestre del año 2020. El trabajo consiste en implementar, en una programa, un algoritmo de fuerza bruta y de programación dinámica para resolver problemas de contenedores y de alineamiento de secuencias. Además, se deberá crear un generador de problemas para que sean utilizados como entrada en la aplicación recién mencionada.

El objetivo de este trabajo consiste en verificar y comparar los tiempos de ejecución de dos de las posibles implementaciones de los problemas anteriormente nombrados. Con esto se busca que los estudiantes sean conscientes de la importancia de los algoritmos con menor tiempo de ejecución y menor gasto de recursos para la resolución de los problemas que deban solucionar en un futuro.

II. Especificación

Los problemas a resolver son el de contenedores (o también llamado mochila); que consiste en buscar el óptimo de combinaciones de elementos para introducir en un depósito y el de alineamiento de secuencias; que trata de alinear dos hileras de caracteres (al añadir espacios es las necesarias) para que sean comparadas como lo más parecidas posibles según ciertas reglas ya predefinidas. A continuación se especifican más detalladamente:

A. Problema de Contenedores

El problema de contenedores es un ejercicio clásico en el Análisis de Algoritmos y en la Investigación de Operaciones, generalmente utilizado para demostrar la gran cantidad de diferentes implementaciones que existen para un mismo problema y analizar su complejidad.

En el presente proyecto, y como en casi cualquier otra descripción, se deberá recibir la capacidad ‘W’ de un contenedor y las especificaciones de ‘N’ objetos distintos que deberán ser agregados al depósito sin sobrepasar la cantidad de este y maximizando el beneficio de agregarlos. Los datos obtenidos por cada elemento serán: su propio peso ‘Wi’, su valor beneficio al ser tomado en cuenta ‘Bi’ y la cantidad en inventario que existe de este ‘Ci’.

B. Solución para los Contenedores

1) ***Fuerza bruta****:* Para realizar la solución al problema de contenedores utilizando la fuerza bruta se hace uso de un algoritmo recursivo que va comprobando cómo resultaría el estado del contenedor al colocar o no colocar un elemento. Para esto se va probando qué sucede si se añade cierto elemento al depósito (y al sumar su valor al beneficio total) y qué sucede si no se añade (no cambiando el beneficio actual), esta comprobación se realiza con todos las combinaciones posibles hasta que lleguen al caso base de que el contenedor se pasó del peso máximo, se cumplió con el peso máximo justo o se llegó al final de la lista de elementos. Para obtener la información de cuales elementos han sido añadidos a la solución final se utiliza una lista que va siendo rellenada por cada una de las recursiones y se recupera de la recursión con la respuesta óptima.

2) ***Programación dinámica****:* En cuanto al algoritmo dinámico para resolver el problema del contenedor se utiliza uno que se apoya en el uso de una matriz para ir comprobando si los elementos pueden ser insertados en el contenedor. La matriz deberá ser de tamaño W (peso máximo de la mochila) por N (número de elementos existentes) y se rellenará de modo que la casilla (i, j) indique el valor beneficio máximo que se ha utilizado para rellenar el depósito cuando es de tamaño j tomando en cuenta todos los objetos hasta el elemento de índice i. Con el objetivo de cumplir lo anterior se deberá comprobar si el nuevo elemento i puede caber en el peso actual y si el resultado de colocarlo significa un incremento en el beneficio.

La utilización de este algoritmo permite identificar fácilmente cuáles fueron los elementos utilizados en la solución final (que será el número encontrado en el último elemento de la matriz) al recorrer la matriz hacia arriba hasta que el siguiente valor ya se haya recorrido, cuando eso sucede se recorre hacia la izquierda hasta que el elemento arriba ya no sea uno previamente visto y se realiza el primer paso de nuevo hasta terminar llegar al inicio de la matriz.

C. Problema de Alineamiento de Secuencias

En cuanto al ejercicio de alineamiento de secuencias, se deberá crear un algoritmo que sea capaz de añadir espacios vacíos en los lugares que sean necesarios de dos hileras de caracteres para maximizar

el resultado de una comparación que busca identificar qué tan iguales son dos secuencias. La semejanza estará definida por la cantidad de aciertos, desaciertos o espacios vacíos encontrados al comparar las dos hileras en la misma posición. Este problema es común al buscar la semejanza entre dos secuencias de bases de ADN.

D. Solución para las Secuencias

1) ***Fuerza bruta****:* Para la solución utilizando la fuerza bruta de este problema se probarán todas las posibles combinaciones que se puedan formar al añadir entre la menor cantidad de espacios necesarios y la mayor de una hilera con las posibles combinaciones de la otra. Para encontrar las combinaciones se recurrirá a una función que realice las permutaciones de la secuencia al añadirle espacios sin desordenar la propia secuencia de caracteres (aprovechándose de las propiedades del lenguaje de programación Python). En la figura 1 se muestran ejemplos de cómo serán las secuencias que se comprobarán.

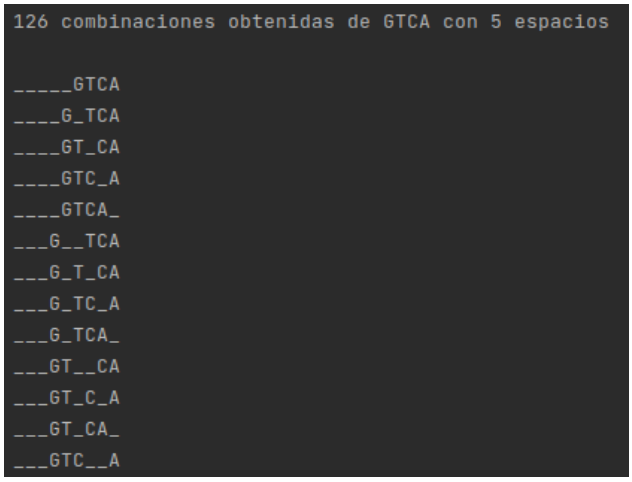

Fig. 1. Ejemplo de combinaciones obtenidas de GTCA con 5 espacios.

2) ***Programación dinámica****:* La solución con programación dinámica se basa en la propuesta para la resolución de este problema dada por Needleman y Wunsch con el "score" 1, -1, -2. Esta solución se basa en crear una matriz de tamaño de la primera hilera por el tamaño de la segunda hilera, una vez creada esta se deben rellenar con valores de 2^i para la primera fila y 2^j los valores de la primera columna. Seguidamente, se deberán calcular los resultados de las celdas (i, j) al elegir el máximo valor de las operaciones con la celda diagonal, izquierda o arriba:

- Diagonal: El valor de la celda diagonal superior izquierda más 1 si el carácter de la secuencia que corresponde a las filas en la posición i es igual al carácter de la otra secuencia en la posición j, de otro modo se le resta 1.

- Izquierda: El valor de la celda a la izquierda del actual menos 2.
- Arriba: El número de la celda de arriba menos 2.

III. Experimentación

A continuación se mostrará el proceso y los resultados de la experimentación realizada para comparar las soluciones en su versión de fuerza bruta contra las de programación dinámica de los problemas antes especificados. Con esto se busca dejar en claro las diferencias en cuanto a complejidad que conlleva el utilizar una implementación sobre la otra. Para la experimentación de cada uno de los problemas se utilizarán 10 problemas generados con un programa capaz de dar un archivo que servirá de entrada para la aplicación capaz de resolver los ejercicios. Los resultados serán mostrados mediante una tabla y un gráfico que permitan ayudar a la comprensión de los mismos. Posteriormente, estos serán analizados para destacar su complejidad temporal.

Hay que tomar en cuenta que para el proceso de experimentación se realizarán las pruebas en un procesador Intel i7 de séptima generación con 8 núcleos que corren a una frecuencia base de 2.9 GHz. Además, para medir los tiempos de los procesos se utilizará la herramienta "time" de Linux que indica cuánto tiempo ha estado en ejecución el proceso en el espacio de usuario, en el espacio del kernel y en total, será este último el que se tomará en cuenta para las mediciones.

A. Experimentación para los Contenedores

Para la realización de las pruebas para los algoritmos sobre el problema de contenedores se crearán 10 casos con cada vez más capacidad para el depósito y cada vez más elementos a tomar en cuenta. Para producir los casos se utilizará el programa generador de este tipo de problemas, aumentando en 5 la cantidad de elementos a insertar en el contenedor, comenzando en 5, y en 10 el peso máximo que soporta el contenedor, iniciando en 100, cada vez que se produce un nuevo archivo.

Los archivos a producir tendrán el nombre "prueba_contenedor_" seguido del número de prueba que les corresponde, además, poseerán la extensión "txt". El peso de los elementos a introducir tendrán las limitaciones de que sus pesos se encontrarán entre 5 y 50, su valor estará entre 20 y 100 y la cantidad que se poseerán de ellos en el inventario siempre será de 1. Ambos algoritmos, el de fuerza bruta y de programación dinámica, serán puestos a prueba sobre los mismos archivos. La figura 2 muestra el resultado de la consola al ejecutar los comandos para generar todos los archivos.

B. Resultados para los Contenedores

Las pruebas efectuadas permitieron la realización de la tabla presente en la figura 3. Esta muestra en los valores de las fila 1 los resultados de los tiempos en segundos del algoritmo de fuerza bruta y en la

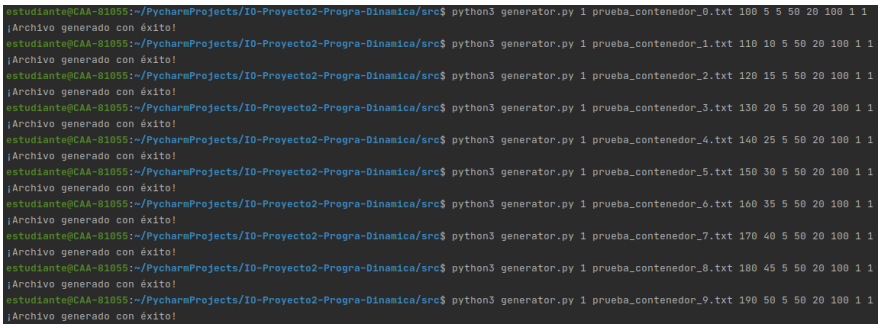

Fig. 2. Comandos utilizados para generar problemas de contenedores.

fila 2, los resultados de las pruebas de programación dinámica. Las columnas muestran el aumento en la complejidad del problema (al aumentar la cantidad de elementos y el peso soportado por el contenedor) y es gracias a estas que se puede analizar y comprobar la gran diferencia entre los tiempos de los dos algoritmos.

Para ayudar a comprender la tabla se realizó la conversión de los tiempos de los últimos 3 resultados del algoritmo de fuerza bruta de únicamente segundos a minutos y segundos, resultando estos en: el problema de 40 elementos se realizó en 1 minuto y 22,644 segundos, la prueba de 45 elementos tardó 1 minuto y 13,746 segundos y el último problema se resolvió en un tiempo de 18 minutos con 29,298 segundos.

Cantidad Contenedor	Elem: 5, Cap: 100	Elem: 10, Cap: 110	Elem: 15, Cap: 120	Elem: 20, Cap: 130	Elem: 25, Cap: 140	Elem: 30, Cap: 150	Elem: 35, Cap: 160	Elem: 40, Cap: 170	Elem: 45, Cap: 180	Elem: 50, Cap: 190
Fuerza Bruta	0,023	0,023	0,032	0,037	0,237	0,946	2,397	82,644	73,746	1.109,298
Dinámico	0,024	0,023	0,025	0,022	0,022	0,024	0,026	0,024	0,026	0,026

Fig. 3. Tabla de tiempos para el problema de contenedores.

Además, se realizó un gráfico de líneas con el objetivo de visualizar de mejor manera el aumento en los tiempos de ambos algoritmos conforme va aumentando la dificultad del problema. La figura 4 muestra el grafo resultante sin ningún tipo de modificación, mientras que en la imagen 5 se puede observar el mismo grafo pero únicamente tomando en cuenta los tiempos menores a 1 segundo.

Según los resultados obtenidos se puede llegar a la conclusión de que el algoritmo de fuerza bruta sigue una función creciente no lineal, esto según la gran cantidad de tiempo que va aumentando en las últimas pruebas, además, puedo confirmar que este algoritmo sigue una función de 2^n (dependiendo directamente de la cantidad de elementos a verificar).

Por otro lado, el algoritmo de programación dinámica sigue una función constante lineal (o al menos

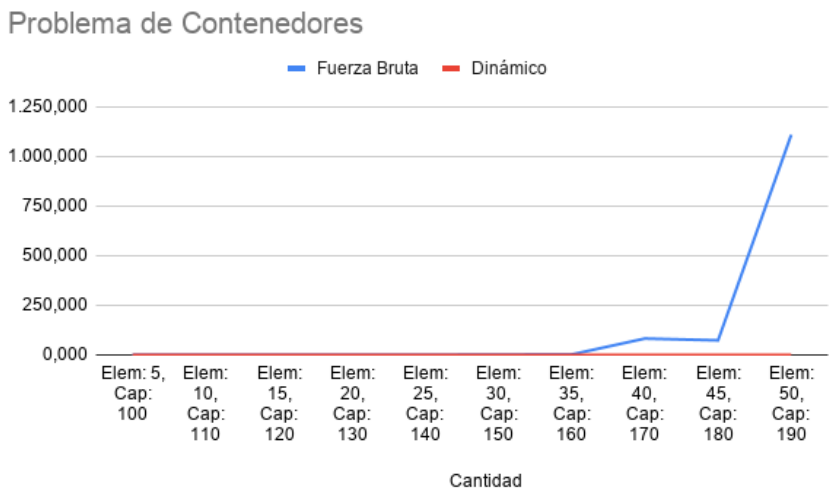

Fig. 4. Grafo resultado de la tabla.

Fig. 5. Grafo resultado únicamente tomando en cuenta los resultados por debajo de 1 segundo.

así demostró ser con las pocas pruebas realizadas). Al analizar el código fuente de este último se puede llegar a la conclusión de que crece de manera lineal siguiendo una función n*m (depende de la cantidad de elementos a agregar al contenedor y de la capacidad máxima del depósito).

C. Experimentación para las Secuencias

Durante la experimentación para los algoritmos solucionadores de alineamientos de secuencias se crearán 10 problemas con cada vez más cantidad de caracteres en las hileras. Para crear los casos a

utilizar se recurrirá a un programa generador de este tipo de problemas, aumentando en 1 el largo posible de una secuencia y en el siguiente archivo aumentando el de la otra (iniciando una hilera en 3 y la segunda en 2).

Los archivos a producir tendrán el nombre “prueba_secuencias_” seguido del número de prueba que les corresponde, además, poseerán la extensión “txt”. Además, ambos algoritmos a probar serán puestos a prueba sobre los mismos archivos. La figura 6 muestra el resultado de consola de utilizar el programa para generar los archivos.

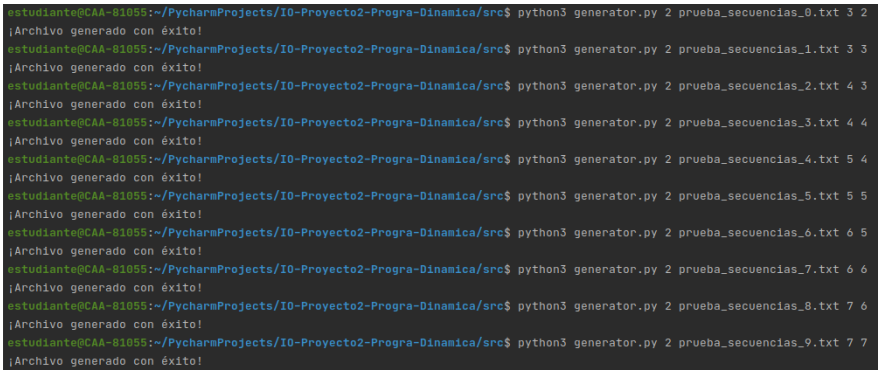

Fig. 6. Comandos utilizados para generar problemas de secuencias.

D. Resultados para las Secuencias

Al finalizar con la ejecución de las pruebas se logró crear una tabla que refleja los resultados de una manera clara y concisa, la cual se encuentra en la figura 7. En esta se muestran los resultados del tiempo de ejecución en segundos del algoritmo de fuerza bruta en la primera fila y los resultados del algoritmo de programación dinámica en la segunda. Las columnas de la tabla permiten verificar el aumento en los tiempos al añadir complejidad a las pruebas, esta dificultad se ve acrecentada en los encabezados de las columnas e indican cúanto es el largo de la hilera 1 (H1) y la longitud de la hilera 2 (H2).

No obstante, al realizar las pruebas de la que sería la columna que representa al problema con un largo de 7 para ambas hileras, el tiempo de espera fue muy largo y se tuvo que detener la ejecución de la misma, por lo que se muestran 9 resultados. Al igual que en el último experimento se realizó la conversión de segundos a minutos de los tiempos más largos en los resultados del algoritmo de fuerza bruta y estos resultaron en que: el problema con longitud 6 duró un total de 3 minutos con 33,671 segundos y el último problema duró un total de 182 minutos con 7,580 segundos.

Además, se realizó un gráfico de líneas con el objetivo de visualizar de mejor manera el aumento en los tiempos de ambos algoritmos conforme va aumentando la dificultad del problema. La figura 4 muestra

Cantidad Alineamiento	H1: 3, H2: 2	H1: 3, H2: 3	H1: 4, H2: 3	H1: 4, H2: 4	H1: 5, H2: 4	H1: 5, H2: 5	H1: 6, H2: 5	H1: 6, H2: 6	H1: 7, H2: 6
Fuerza Bruta	0,023	0,037	0,026	0,075	0,198	1,610	16,881	213,671	10.927,580
Dinámico	0,016	0,023	0,023	0,023	0,023	0,023	0,022	0,023	0,023

Fig. 7. Tabla de tiempos para el problema de alineamiento de secuencias.

el grafo resultante sin ningún tipo de modificación, mientras que en la imagen 5 se puede observar el mismo grafo pero únicamente tomando en cuenta los tiempos menores a 1 segundo.

El gráfico de líneas para lograr una mejor comprensión de los resultados se puede observar en la figura 8. Además, la figura 9 muestra el mismo grafo pero solamente tomando en cuenta los resultados con un tiempo menor a 1 segundo.

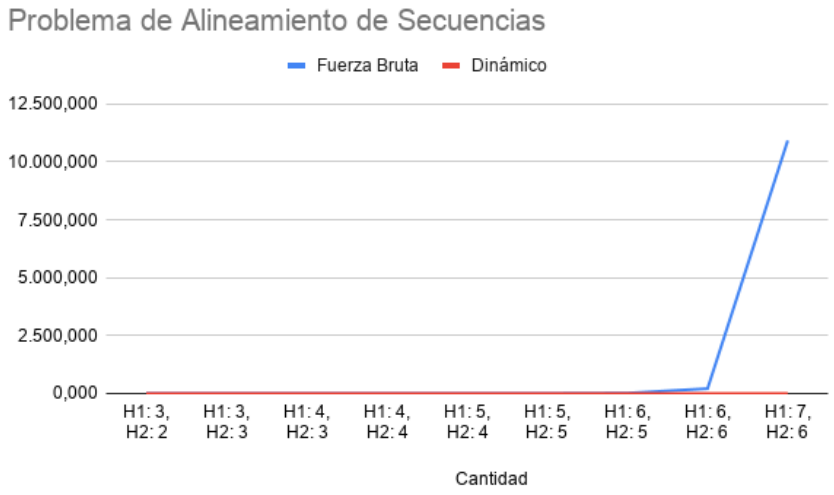

Fig. 8. Grafo resultado de la tabla.

Los resultados obtenidos indican que los tiempos resultado del algoritmo de fuerza bruta crecen de manera muy rápida siguiendo una función no lineal. Considero que este gran aumento se debe principalmente a la gran cantidad de combinaciones que son tomadas en cuenta a la hora de buscar un resultado, ya que las combinaciones del algoritmo están dadas por el factorial del largo de la hilera. Al observar el código, se puede llegar a la conclusión de que la complejidad temporal del programa sigue la función n*n!*n! (siendo el primer n el menor largo de las hileras, mientras que la segunda y tercera variable corresponde a la simplificación de la fórmula para hallar la cantidad de combinaciones de n elementos).

En cuanto al programa que utiliza programación dinámica, se puede observar que sus tiempos son muy reducidos y se mantienen casi constantes al aumentar la complejidad de los problemas, por lo que

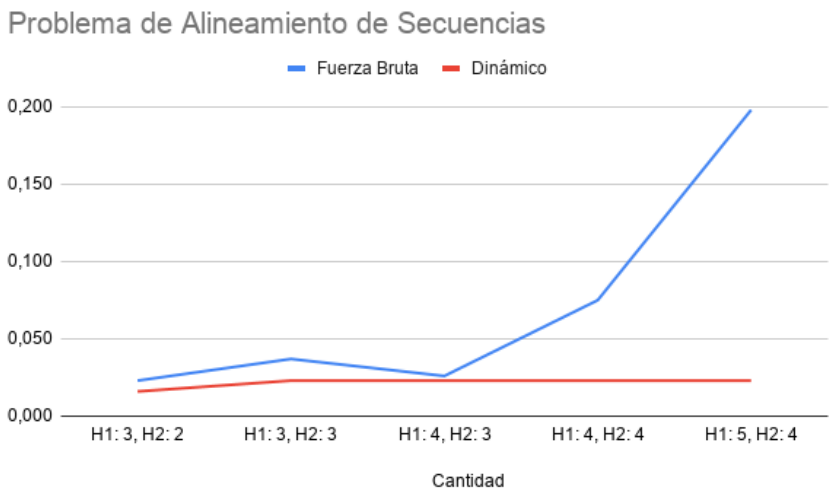

Fig. 9. Grafo resultado únicamente tomando en cuenta los resultados por debajo de 1 segundo.

se puede concluir que sigue una función lineal con una pendiente muy pequeña. Al analizar el código fuente del programa se puede llegar a la conclusión de que se sigue una función de n*m, siendo n el largo de la primera secuencia y m el largo de la segunda.

IV. Conclusión

Para finalizar, deseo recalcar la importancia de las implementaciones que utilizan programación dinámica para realmente hacer factibles el uso de computadores para resolver problemas verdaderamente complejos. Lo anterior afirmado por los experimentos desarrollados durante este proyecto, los cuales demostraron que existe una gran diferencia entre los tiempos de ejecución manejados. Todo esto tomando en cuenta que los experimentos realizados únicamente aumentaban en pequeños valores la cantidad de datos a procesar, y que en un proceso profesional el tamaño de estas entradas puede ser aumentado en una magnitud de miles.

Además, es posible destacar la gran variedad de aplicaciones que posee la programación dinámica para resolver cualquier cantidad de problemas, únicamente es necesario poseer un pensamiento capaz de detectar y dividir en pequeños subproblemas uno mayor e ir siempre buscando el camino óptimo. Debido a la gran mejora que representa en un sistema la utilización de un algoritmo de este tipo, considero necesario para cualquier profesional en el área de la computación tener siempre en mente la posibilidad de emplear un programa de programación dinámica para crear sistemas que realmente marquen la diferencia.