Diseño y Análisis de Algoritmos: Solución problema B del proyecto

Jaime Carvajal M 201632567, Nicole Bahamon M 201629594

16 de diciembre de 2020

Algoritmo de solución

Al ser tan similar el problema B al descrito en el capitulo 15 del libro de Corman[1], desde un principio supimos que se debía utilizar programación dinámica de alguna manera. Teniendo en cuenta los números que se nos daban (Longitud de la varilla, posición de cada corte), se decidió intentar llenar un arreglo con todas las posibles combinaciones de cortes con las respuestas de cada una, para así poder encontrar las soluciones más rápidamente. Este era el algoritmo original que teníamos, pero intentando encontrar maneras de incrementar la eficiencia del código, descubrimos la optimización de Knuth[2]. Esta optimización reduce drásticamente la complejidad temporal, aunque incrementa un poco la complejidad espacial, para problemas similares al que nos enfrentamos en esta situación.
Nuestro algoritmo se puede dividir en tres partes. La primera es la inicialización de todas las variables necesarias, la segunda es llenar el arreglo de posibles soluciones con ayuda de la optimización de Knuth, y la tercera es devolver el valor final. Para hacer las cosas más fáciles, se hizo una tabla de todos los valores que se utilizaron.

E/S/C	Nombre	Tipo	Descripción
E	len	Int	El tamaño de la barra a cortar
E	cortes	Int[]	Las posiciones en las que se debe cortar la barra
S	costo	Int	El menor costo para cortar la barra en todas las posiciones especificadas
C	resps	Int[][]	El arreglo de respuestas anteriores utilizado para llevar a cabo programación dinámica
C	mid	Int[][]	Un arreglo importante para la optimización de Knuth
C	neo	Int	Nuevo tamaño de arreglos
C	pos	Int[]	Arreglo de cortes junto con valores agregados.

La primera parte anteriormente mencionada requiere antes que se entiendan las precondiciones del problema, por lo que estas se explican a continuación.
Precondición:
len(cortes) $\leq len \wedge (|0 \leq i < len(cortes) : 0 < cortes[i] < len) \wedge len \leq 100 \wedge (j|0 \leq i < j < len(cortes) : cortes[i] < cortes[j])$
Lo que esta precondición quiere decir es que no puede haber una mayor cantidad de cortes de lo que la barra es de larga, no se puede hacer cortes en las esquinas de la barra ni que tengan

un valor superior a la longitud de esta, y la longitud máxima de la barra es de 100. Además, no puede haber valores de cortes repetidos y estos deben estar en orden ascendente.
Para la primera parte, se debe llevar a cabo la inicialización de todos los valores que se denominan C en la tabla (Por creados). Esto quiere decir que se debe inicializar neo, resps y mid. Neo es el nuevo tamaño de los próximos arreglos a ser creados, por lo que debe ser el primero en inicializarse. Este nuevo arreglo tiene tamaño de 2 más que el arreglo de cortes, esto con la idea de agregar los valores 0 al inicio y len al final de este, mientras que el resto de los valores se mantienen iguales. En un futuro, esto ayudará a llevar a cabo el procedimiento para guardar respuestas. Resps tiene un nombre que explica completamente para qué está hecho. Es un arreglo de arreglos de números, de tamaño neoxneo, que tiene la función de guardar las respuestas calculadas de cada valor, comenzando desde 0 y acabando en 20, pero cambiando los valores entre estos para cada fila del arreglo. Mid se inicializa para ser del mismo tamaño que resps, y se utilizará para agilizar el algoritmo lo más posible.
La postcondición de esta parte del algoritmo se explica a continuación:
neo = 2 + len(cortes) $\wedge pos[0] = 0 \wedge pos[neo - 1] = len \wedge len(resps) = neo \wedge len(resps[0]) = neo \wedge len(mid) = neo \wedge len(mid[0]) = neo$
Esta también funciona como la precondición para la segunda parte del método, en la que se llena la tabla de resps utilizando la optimización de Knuth. Se va llenando los valores de resps con los resultados de hacer diferentes cortes en diferentes ordenes, y se va llenando mid con la posición en la que ocurre el valor mínimo, para así evitar tener que buscarlo y ahorrarse un loop. Por último, se devuelve el valor que se encuentra en [0, neo - 1] en resps, ya que este es el valor para el precio optimizado.

Análisis de complejidades espacial y temporal

La complejidad espacial se mide teniendo en cuenta que los ints valen 4 bytes cada uno, por lo que se puede decir que, teniendo en cuenta que se crean dos arreglos de arreglos de ints y un arrelgo de ints, todos de dimensiones neo, se puede decir que la complejidad espacial tiene una ecuación de 4(2(neoxneo)+neo), lo que indica que crece de forma cuadrática y se puede denominar complejidad espacial cuadrática.
La complejidad temporal se mide específicamente teniendo en cuenta los valores temporales de cada acción y la cantidad de veces que se repite. Sin embargo, la optimización de Knuth se lleva a cabo para asegurar complejidad temporal de $O(n^2), porloquesepuedeasumirqueesteeselcaso.$

Comentarios finales

Este es probablemente el algoritmo más eficaz que llevamos a cabo. Todas las pruebas se corrían rápidamente sin importar la cantidad de cortes o la longitud de la barra. La utilización de la optimización de Knuth también ayudó a reducir tiempos que ya eran cortos para empezar.

referencias

1. Cormen, et al. Ïntroduction to Algorithms, Third Edition", MIT Press, revisado el 10 de diciembre, 2020.

2. Anónimo, "Knuth's Optimization", AlgoWiki, obtenido de https://wiki.algo.is/Knuth%27s%20optimization el 12 de diciembre de 2020.