Práctica 3: Algoritmo Voraz y Programación Dinámica

Julio José Reyes Hurtado
Antonio García Castillo

May 20, 2018

1 Introducción

En esta memoria se expone la utilización de varias técnicas de algorítmica, como es la técnica voraz o greedy y la técnica de programación dinámica.

2 Técnica Voraz: Suma hasta un número

Este problema plantea que mediante una serie de dígitos determinados podamos alcanzar mediante la suma un número que nosotros elijamos y si la suma no es alcanzable nos retornara los dígitos que sumen la cantidad mas cercana.

2.1 Datos del problema

Se nos dará un conjunto con números y mediante estos números deberemos llegar a un número m mediante la suma de los elementos del cojunto, en caso de que no existiera se dará la suma inferior más cercana a m.

2.2 Representación de la solución

Una solución S =(x_1,x_2...,x_n) con $0 < x_i <=$ m donde x_i es el número que podemos sumar o no a la solución, ya que al menos debe ser mayor que cero y menor o igual que la solución a alcanzar.

2.3 Formulación matemática

Max(candidatos) si candidato + suma <= m.

2.4 Candidatos

El conjunto candidatos serán los valores de los cuales tendremos que intentar llegar a m.

2.5 Función solución

Se comprueba que el conjunto solución es una solución.

2.6 Función selección

Se elige el del elemento más grande en el vector de candidatos, si su valor mas el de la suma que llevamos es mayor que el elemento descartamos ese candidato, sino se añade al vector de resultado.

2.7 Función factible

Si añadimos un elemento nuevo a partir de este conjunto resultado podemos seguir obteniendo la solución ya que si la suma del elemento nuevo + lo que se encuentran en el cojunto resultado es mayor que el valor de m entonces descartamos ese valor.

2.8 Función objetivo

Comparamos si el valor almacenado en el indice del vector + la suma total es mayor que m entonces se descarta el valor del vector ordenado y no es mayor se añade al conjunto solución.

2.9 Eficiencia Teórica

Partimos de dos funciones principales.
Con la función sort ordenamos el vector de menor a mayor y vamos cogiendo el mas grande, esta función implementada en el TDA vector que proporciona la STL tiene una eficiencia de n*log(n), por otro lado el bucle que realiza las operaciones en su peor caso, que no llegue a obtener la suma, recorrerá el vector por completo por lo que tendremos una eficiencia de O(n).

Como las dos funciones no estan anidadas la eficiencia será la suma de ambas, pero utilizando la notación Ogrande la eficiencia sería de O(n*log(n)). Con la función sort evitamos realizar una llamada en cada iteración que resultaría una eficiencia de $O(n^2)$, aumentando la eficiencia del algoritmo.

2.10 Dificultad

En este primer primer ejercicio la dificultad que nos ha presentado desde 0 muy fácil a 10 muy difícil a nuestro parecer ha sido de 2.

3 Programación Dinámica: Suma hasta un número

A continuación se presenta el problema anteriormente presentado pero resolviendolo mediante la técnica de programación dinámica.

3.1 Principio de optimalidad de Bellman

Para saber si en un problema podemos obtener la solución óptima de manera que las sumas de las soluciones de sus subproblemas sean óptimas, es decir, cualquier subsecuencia de una secuencia óptima debe ser óptima. Como para este problema que se cumpla la optimalidad es obtener el resultado, partimos de una secuencia de candidatos = (x_1,x_2,x_3) y una m = k. Si con alguna de las combinaciones de de la secuencia de candidatos conseguimos conseguiremos la solución óptima.

3.2 Ecuación recurrente

$$Sel(m,k) = \begin{cases} 0 & si \quad m = 0 \parallel k = 0 \\ inf & si \quad m < 0 \parallel k < 0 \\ max(Suma(k-1, m-k) + k, Suma(k-1, m)) & \forall \quad k > 0, m > 0 \end{cases} \tag{1}$$

3.3 Estrategia de aplicación

Se comienza rellenando la posicion 0 de todas las filas con true ya que sabemos que para conseguir el número cero cualquier número es válido, bastaría con no echarlo. Rellenamos todas las columnas de la primera fila a cero. La matriz se compone de tantas filas como elementos candidatos tengamos y de tantas columnas como el elemento que queremos alcanzar sumandole 1. Para rellenar los huecos restantes se rellenarán con true o false dependiendo de si el valor de la columna menos el valor de la fila-1 del vector de candidadtos es mayor o igual a cero el hueco de fila y columna se rellena con cero si ninguna condición de las expuestas en el código se cumplen y con uno en los demás casos. Si es menor que cero entonces se rellenara con el elemento que hay en su misma columna pero en la fila anterior con esto sabemos que el elemento a coger no sería el de esa fila sino el de la anterior.

3.4 Recuperación de la solución

Para recuperar la solución debemos situarnosen el hueco en la última fila de la última columna y mientras que no lleguemos a la columna cero seguiremos aplicando lo siguiente: Si el elemento en la posición i,j es diferente a la de su i-1(fila anterior) entonces echaremos el número de candidatos que está en esa fila, bajaremos una fila y restaremos el candidato a la columna actual. Si no disminuiremos la fila.

3.5 Eficiencia

Para los problemas que se resuelven con programación dínamica la como estamos manejando tablas la eficiencia será de m x n, concretamente en nuestro código utilizamos dos for anidados para rellenar la matriz de tamaño n x m y otros dos para imprimirla, esto es el mayor coste que tiene por lo que podemos decir que la eficiencia es O(nxm).

4 Programación dinámica: Operaciones básicas

En este problema deberos realizar un programa que nos diga las operaciones que se realizan para alcanzar una cifra.

4.1 Ecuación recurrente

$$S(m,k) = \begin{cases} 0 & si \quad m = 0 \parallel k = 0 \\ -inf & si \quad m < 0 \parallel k < 0 \\ -inf & si \quad k > th \\ min(A(k-1,M), A(k-1,m-k), \\ A(k-1,m+k), A(k-1,m/k), A(k-1,m*k)) & \forall \quad k > 0, m > 0 \end{cases} \tag{2}$$

4.2 Estrategia de aplicación

Para este problema las dimensiones de la matriz tendrá un número de columnas igual al máximo de los operandos disponibles multiplicado por el objetivo más uno, así como las filas serán el número de operandos más uno. Para rellenar la matriz la primera fila la rellenaremos con los valores desde cero hasta el número de columnas existentes, la primera columna se rellena todo con ceros ya que podemos conseguirlo al multiplicar cualquiera de los operandos por cero. Para rellenar la matriz se tiene que elegir el valor mínimo entre coger el valor de la fila anterior, o realizar las operaciones entre el valor a conseguir y el operando que estamos utilizando.

4.3 Recuperación de la solución

Para conseguir la solución desde la última posición de la matriz debemos compararlo con el de la fila anterior, si vemos que es igual entonces no cogemos el operando que esta en esa fila y vamos a la fila anterior realizando la operación indicada al valor de la columna en la que estamos con el número que no se ha echado, si en la comparación son diferentes, echamos el valor de esa fila y le aplicamos la operación al valor de la columna y bajamos la fila para seguir tratando el siguiente operando.

4.4 Eficiencia

Al ser un caso de programacion dinámica podemos presumir que tendrá una eficiencia en el peor caso del tamaño de la matriz, es decir, nxm para este caso m será el número de columnas de la matriz que es el valor a alcanzar por el máximo de los valores de n. Viendo el código podemos confirmar esta presunción ya que podemos ver varios bucles de tamaño de las filas y las columnas pero podemos ver que cuando la rellenamos utilizamos un bucle n que recorre las filas y otro de tamaño m para las columnas, así como para imprimirla se utilizan los mismos bucles. Para buscar

la solución utilizamos un solo bucle por lo que podemos decir que la eficiencia en el peor caso sería de O(nxm).