Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2283 - Diseño y Análisis de Algoritmos
Profesor: Nicolás Van Sint Jan
Ayudante: Dante Pinto

Ayudantía 5

Algoritmos Codiciosos

Problema 1

Demuestre que dada una función de frecuencias relativas $f : \Sigma \to (0, 1)$, el algoritmo de Huffman encuentra una Σ-codificación, τ, libre de prefijos que minimiza la función $lp_f(\tau)$.

Solución: La solución a este problema se encuentra en las slides de la clase 8 (algoritmos codicioso 2 - electric boogaloo) y la grabación de su desarrollo está en este link:

`https://zoom.us/rec/share/pnZyW7obmOWyEbntwNXo-ejGX8bU9j4z1DsUt_xzjmpuFZuNvFdQKX_dM9QfsdPh.bOiwxjY3OCWS6kPL?startTime=1662752373000`

Problema 2

Sean A un arreglos de números naturales de largo n. Definimos la operación $\texttt{Cmp}(A, i, j)$, con $0 \leq i < j \leq n$, como la compresión de elementos del arreglo, que consiste en tomar el subarreglo $A_{ij} = [a_i, a_{i+1}, ..., a_j]$, sumar sus elementos y luego reemplazar el subarreglo, dentro de A, por esta suma.

Por ejemplo, sea $A = [1, 10, 100, 1000, 10000]$, tendremos que $\texttt{Cmp}(A, 1, 3) = [1, 1110, 10000]$, $\texttt{Cmp}(A, 0, 1) = [11, 100, 1000, 10000]$ y $\texttt{Cmp}(A, 2, 4) = [1, 10, 11100]$.

Dado dos arreglos A y B, queremos averiguar si es posible comprimir los arreglos hasta que ambos sean iguales[1] y, si es el caso, encontrar el largo de máximo para el que esto es posible.

Entregue un algoritmo que dado dos arreglos A y B, de largos n y m respectivamente, retorne el largo máximo para el que los arreglos son iguales luego de comprimirlos una cantidad arbitraria de veces. Analice la complejidad del algoritmo.

Solución: Se dio una intuición de la solución en el video del Problema 1 (Desde 1:04:33), pero a continuación se encuentra una solución formal.

Una algoritmo codicioso que resuelve este problema es el siguiente:

[1] Decimos que dos arreglos son iguales si tienen el mismo largo y son iguales elemento a elemento.

```
Function maxlength(A, B)
    if sum(A) ≠ sum(B) then
        return 0
    length = 0
    i = 0
    j = 0
    while i < n do
        S_A = A[i]
        S_B = B[j]
        while S_A ≠ S_B do
            if S_A < S_B then
                i++
                S_A += A[i]
            else
                j++
                S_B += B[j]
        length++; i++; j++
    return length
```

Es claro que el algoritmo entrega un posible largo tal que los arreglos comprimidos serán iguales, pues lo que va haciendo al sumar S_A y S_B es básicamente comprimir elemento por elemtno hasta que ambos son iguales, para luego avanzar ambos y continuar iterando.

Por lo anterior, lo que queremos demostrar es que el algoritmo retorna el largo máximo para el arreglo A tal que $A = B$ luego de comprimirlos. Observando la ejecución del algoritmo, podemos notar que haremos una compresión a ambos arreglos cuando se cumpla que $S_A = S_B$, que es cuando se aumenta *length*, por lo que observando A al final de la ejecución, podemos abstraernos de las compresiones y mirar solamente los índices en lo que vamos a comprimir.

Sean $i_1, i_2, ..., i_x$ y $j_1, j_2, ..., j_x$ las secuencias de índices donde debemos comprimir A y B según el algoritmo. Es claro que si sumamos los elementos de A entre cada par de índices, esta suma será igual a la suma de los elementos de B entre los índices análogos, pues dichas sumas serán los elementos de los arreglos finales y ambos deben ser iguales.

Luego, podemos demostrar lo que buscamos usando contradicción; suponga que $I = i_1, i_2, ..., i_x$ y $J = j_1, j_2, ..., j_x$ no son las secuencias óptimas de compresión, es decir, existen dos secuencias de compresión, $K = k_1, k_2, ..., k_y$ y $L = l_1, l_2, ..., l_y$, tales que K, L son secuencias óptimas y, además, producen arreglos más largos que I, J.

Si K es mejor que I, sabemos que se debe cumplir que $y > x$, pues más índices significa que hay menos compresiones (en el mejor de los casos $x = n$, lo que significa que todos los segmentos del arreglo tienen tamaño 1 y, por lo tanto, no hay compresiones) lo que lleva a un arreglo más largo.

Como ambas secuencias son diferentes, deben haber al menos dos índices i_d, k_d tales que $i_d \neq k_d$ y que además cumplen con que $i_d < k_d$, pues `maxlength` comprime/corta el arreglo en cuanto encuentra una igualdad.

Por otra parte, tenemos que $sum(A[i_{d-1}, i_d]) = sum(B[j_{d-1}, j_d])$ y además $sum(A[k_{d-1}, k_d]) = sum(B[l_{d-1}, l_d])$, sin embargo, como i_d y k_d son los primeros índices donde difieren las secuencias, sabemos que $i_{d-1} = k_{d-1}$, lo que significa a su vez que:

$$
\begin{aligned}
sum(A[k_{d-1}, k_d]) &= sum(A[i_{d-1}, k_d]) \\
&= sum(A[i_{d-1}, i_d]) + sum(A[i_d + 1, k_d])
\end{aligned}
$$

Análogamente:

$$\begin{aligned} sum(B[l_{d-1}, l_d]) &= sum(B[j_{d-1}, l_d]) \\ &= sum(B[j_{d-1}, j_d]) + sum(B[j_d + 1, l_d]) \end{aligned}$$

Luego, como $sum(A[i_{d-1}, i_d]) = sum(B[j_{d-1}, j_d])$, podemos deducir de las dos ecuaciones anteriores que $sum(A[i_d + 1, k_d]) = sum(B[j_d + 1, l_d])$

Siguiendo esta idea, podremos subdividir las sumas de $A[k_{d-1}, k_d]$ y $B[l_{d-1}, l_d]$ en dos, creando nuevas secuencias de índices $K' = k_1, k_2, ..., k_{d-1}, i_d, k_d, ..., k_y$ y $L = l_1, l_2, ..., l_{d-1}, j_d, l_d, ..., l_y$.

Dado que encontramos nuevas secuencias que generan arreglos más largos, hemos llegado a una contradicción, por lo que nuestra hipótesis original "Existen secuencias óptimas, K, L, que producen arreglos más largos que I, J" es falsa, lo que quiere I, J deberán ser las secuencias óptimas y, por lo tanto, nuestro algoritmo encontrará el mayor largo posible para los arreglos.