Avanze del proyecto

Análisis y Diseño de Algoritmos(ADA)

jorge.mayna

Junio 2020

repo : https://github.com/jorgemayna/ADA_2020_proyect

1. Introducción

Los 2 arreglos de 1s y 0s que recibimos los guardamos como strings y para hallar sus respectivos bloques usamos el siguiente algoritmo:

```
BLOQUES(string s, tamaño t)
1: for i = 1 to t
2:    if s[i] = '1'
3:       first = i;
4:       while s[i] = '1'
            i++
5:       R.push_back(first, i - 1, i - first+1)
6: return R
```

El algoritmo no solo te retorna cada bloque sino que también su tamaño lo que sera usado posteriormente. Los bloques son guardados en los arreglos A[n] y B[m].

2. Algoritmo voraz

Para el algoritmo voraz se optó por el acercamiento mas sencillo, por lo cual no retorna un matching con peso mínimo, pero si retorna un matching valido. Para facilitar las cosas asumiremos que la longitud de A[n] es mayor que la de B[m](En el código en c++ si se valida esto) y que cuando usemos la notación A[i] nos estamos refiriendo al modulo del bloque i en vez de al bloque en si mismo.

GREEDY($Bloques\ A, Bloques\ B$)	*cost*	*times*
1: cont = 0	c_1	1
2: **for** $i = 1$ to $B.size$	c_2	$m + 1$
3: matching.push_back(i,i)	c_3	m

4:	cont = cont + A[i] / B[i]	c_4	m
5:	**for** $i = B.size + 1$ to $A.size$	c_5	$n - m + 1$
6:	matching.push_back(i,B.size)	c_6	$n - m$
7:	cont = cont + A[i] / B[B.size]	c_7	$n - m$
8:	return (matching,cont)		

.

En el algoritmo anterior simplemente se separo las operaciones en 2 ciclos "for"para evitar comprobaciones innecesarias usando solo 1.
Sea $T(n)$ el tiempo de ejecución con el tamaño de A como n:
$T(n) = c_1 + c_2(m+1) + c_3(m) + c_4(m) + c_5(n-m+1) + c_6(n-m) + c_7(n-m)$
$T(n) = n(c_5 + c_6 + c_7) + m(c_2 + c_3 + c_4 - c_5 - c_6 - c_7) + c_1 + c_2 + c_5$
Si le despreciamos las diferencias entre constantes:
$T(n) = n(3c) + 3c = O(n)$

3. Recurrencia

Siendo A[n] y B[m] los vectores con los bloques de los 2 arrays de 1s y 0s originales, usaremos la notación A[i] para referirnos al modulo del bloque i en vez de al bloque en si mismo.
NOTA : Para mejor entendimiento separaremos las partes para evitar aglomeración.

$$\begin{aligned} UNO &= Min_{z=1}^{z=j-2}(OPT(i-1,z) + A[i]/\sum_{x=z+1}^{x=j} B[x]) \\ DOS &= OPT(i-1,j-1) + A[i]/B[j] \\ TRES &= Min_{z=1}^{z=i-2}(OPT(z,j-1) + (\sum_{x=z+1}^{x=j} A[x])/B[j]) \end{aligned}$$

$$OPT(i,j) = \begin{cases} A[i]/\sum_{z=1}^{z=j} B[z] & i = 1 \\ (\sum_{z=1}^{z=i} A[z])/B[j] & j = 1 \\ Min(UNO, DOS, TRES) & i > 1 \text{ and } j > 1 \end{cases}$$

4. Algoritmo recursivo

Para resolver el problema de manera recursiva se optó por el siguiente algoritmo:

OPT(i,j)
1: **if** i = 0 . .
2: cont = SUM(B[i], $1 \leq$ i $\leq$ j)
3: Return A[i]/cont
4: **if** j = 0

5: cont = SUM(A[i], $1 \leq i \leq i$)
6: Return cont/[j]

5. Algoritmo memoizado

Para realizar el memoizado lo único que se agrego fue una matriz nxm que guardaba los OPT ya calculados para usarlos posteriormente si se necesitaba.