Tarea 2 Problemas conceptuales

Iván David Valderrama Corredor
Ingeniería de Sistemas y Ciencias de la Computación
Pontificia Universidad Javeriana, Cali

15 de febrero de 2019

Índice

1. Problemas conceptuales

1.1. Problema 15-4:Printing Neatly(Cormen et. al. página 405).

*El texto tiene n palabras, cada un con longitud de caracteres variable l1,l2,l3
*Entre palabras hay un solo espacio
*Cada linea tiene un maximo de M caracteres
*i y j son las palabras que se intentan probar por linea
* La formula $M - j + i - \sum_{k=i}^{j} lk$ nos da el espacio sobrente al final de cada linea

Proporcione un algoritmo de programación dinámica para imprimir un párrafo de n palabras de forma ordenada en una impresora.

$$Lc(i,j) = \begin{cases} INF & \text{extras[i,j] <(i.e., palabra i,...,j no encaja),} \\ 0 & \text{j = n and extras[i,j] >= 0 (el costo de la ultima linea seria 0),} \\ (extras[i,j])^2 & \text{de otra forma} \end{cases}$$

Respuesta

```
INF = 99999999999

def printingNeatly (l, M):
    n = len(l)
    extras = [[0 for i in range(n)] for i in range(n)]

    # lc[i][j] es la matriz que tiene los costos
    lc = [[0 for i in range(n)] for i in range(n)]

    # c[i] tendra los costos totales del arreglo de palbras desde 0 hasta i
    c = [0 for i in range(n)]

    # p[] p es el arreglo usado para impimir el resultado
    p = [0 for i in range(n)]

    # indica los espacios adicionales, si las palabras de la palabra i y j s
    for i in range(n):
        extras[i][i] = M - l[i]
        for j in range(i + 1, n):
            extras[i][j] = (extras[i][j - 1] -
                                l[j - 1] - 1)
```

```
    # Calcula el costo de linea
    for i in range(n):
        for j in range(i, n):
            if extras[i][j] < 0:
                lc[i][j] = INF;
            elif j == n and extras[i][j] >= 0:
                lc[i][j] = 0
            else:
                lc[i][j] = (extras[i][j] *
                            extras[i][j])

    # Calcula el minimo costo del arreglo
    c[0] = 0
    for j in range(1, n):
        c[j] = INF
        for i in range(1, j):
            if ((c[i - 1] + lc[i][j]) < c[j]):
                c[j] = c[i-1] + lc[i][j]
                p[j] = i
    return(p, c)
```

[1] [2]

Analice los requisitos de tiempo y espacio de ejecución de su algoritmo.

Tanto el tiempo como el espacio de ejecución del algoritmo son $\Theta(n^2)$

1.2. Problema 15-9:Breaking a String(Cormen et. al. página 410).

Diseñe un algoritmo que, dados los números de caracteres después de los cuales se rompa, determine una manera menos costosa de secuenciar esos descansos. Más formalmente, dada una cadena S con n caracteres y una matriz L (1..m) que contiene los puntos de interrupción, calcula el costo más bajo para una secuencia de interrupciones, junto con una secuencia de interrupciones que permite este costo

El problema es muy similar al problema de "matrix-chain multiplication"

Respuesta

```
INF = 99999999999

def Sequencebreaks(L, TablaCorte, i, j):
    if (j - i >= 2):
        print ("Break at",L[k])
        Sequencebreaks(L, TablaCorte, i, k)
        Sequencebreaks(L, TablaCorte, k, j)

def BreakingString(n, L):
    #Ej: L:[20 17 14 11 25]
    L= L.sort()
    L.insert(1, 0)
    L.append(n)
    # Ej: L:[0 20 17 14 11 2 5 n]
    m = len(L)
    #Nuevas tablas
    TablaCosto = [[1 for i in range(m)] for i in range(m)]
    TablaCorte = [[1 for i in range(m)] for i in range(m)]
    for i in range(1,len(m)):
        TablaCosto[i, i],TablaCosto[i, i + 1] = 0, 0
    TablaCosto[m, m] = 0
    for lon in range(3,m):
        for i in range(1,m - lon + 1)
            j = i + lon - 1
            TablaCosto[i, j] = INF
            for k in range(i + 1, j - 1)
                if (TablaCosto[i, k] + TablaCosto[k, j] < TablaCosto[i, j]):
                    TablaCosto[i, j] = TablaCosto[i, k] + TablaCosto[k, j]
                    TablaCorte[i, j] = k
            TablaCosto[i, j] = TablaCosto[i, j] + L[j] - L[i]
    print ("The minimum cost of breaking the string is", TablaCosto[1, m])
    Sequencebreaks(L, TablaCorte, 1, m)
```

[3]

Dada cada iteración del bucle, mas los bucles internos, el tiempo total de ejecución es $\Theta(m^3)$

1.3. Ejercicio 9:High Performance Computing(Kleinberg and Tardos página 320).

Respuesta

(a)

Día	[1]	[2]	[3]	[4]	[5]	[6]
X	[10]	[9]	[8]	[7]	[9]	[8]
S	[9]	[8]	[7]	[6]	[3]	[1]

La mejor solucion sería hacer un reboot el día 4 debido a que el total seria $9+8+7+0+9+8=41$ y sin hacer reboot seria $8+6+4+2=34$.

(b)

```
def HighPerformanceComputing(n,X,S):
        mini = min(X)
        sumi=0
        j=0
        for i in range(0,n):
                if(X[i]==mini):
                        j=i
                if(X[i]>=S[i-j]):
                        sumi+=S[i-j]
                else:
                        sumi += S[i-j]-X[i]
        return(sumi)
```

El tiempo total de ejecución es $\Theta(n)$

Referencias

[1] CLRS Solutions, *15-4 Printing neatly.*
https://walkccc.github.io/CLRS/Chap15/Problems/15-4/

[2] Tushar Roy, *Text Justification Dynamic Programming.*
https://www.youtube.com/watch?v=RORuwHiblPc

[3] CLRS Solutions, *15-4 Printing neatly.*
https://walkccc.github.io/CLRS/Chap15/Problems/15-9/

[4] Computer Science, *Using dynamic programming to maximize work done.*
https://cs.stackexchange.com/questions/48980/using-dynamic-programming-to-maximize-work-done