IIC2283 — Diseño y Análisis de Algoritmos
Ayudantía 5

Problema 1

Dados dos arreglos A y B, con B de largo m, decimos que B es un *subarreglo contiguo* de A, si existe δ tal que para todo i entre 0 y $m-1$, $A[i+\delta]$ está bien definido, y $A[i+\delta] = B[i]$.

Desarrolle un algoritmo que dado un arreglo de enteros de largo n, retorne un subarreglo contiguo de suma máxima.

Idea de solución El algoritmo funciona en base a dividir para conquistar.

function SM($A[1, \ldots, m]$)
  if $m = 1$ **then return** A
  end if
  $L \leftarrow$SM($A[1, \ldots, \lfloor m/2 \rfloor]$)
  $R \leftarrow$SM($A[\lfloor m/2 \rfloor + 1, \ldots, m]$)
  $idx_R \leftarrow \text{argmax}_j \left[\sum_{i=\lfloor m/2 \rfloor + 1}^{j} A[i]\right]$ ▷ j entre $\lfloor m/2 \rfloor + 1$ y m
  $idx_L \leftarrow \text{argmax}_j \left[\sum_{i=j}^{\lfloor m/2 \rfloor} A[i]\right]$ ▷ j entre 1 y $\lfloor m/2 \rfloor$
  return Arreglo de suma máxima entre L, R, y $A[idx_L, \ldots, idx_R]$
end function

Ahí, L representa al arreglo de suma máxima contenido en la primera mitad, R al arreglo de suma máxima contenido en la segunda mitad, y $A[idx_L, \ldots, idx_R]$ al arreglo de suma máxima que contiene tanto a $A[\lfloor m/2 \rfloor]$ como a $A[\lfloor m/2 \rfloor + 1]$.

La corrección del algoritmo se obtiene por inducción, considerando que esos tres casos son los únicos posibles. El tiempo de ejecución se obtiene de la siguiente recurrencia:

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T\left\lfloor \frac{n}{2} \right\rfloor + kn & n > 0 \end{cases}$$

que pertenece a $\Theta(n \log n)$.

Problema 2

Dos jugadores buscan repartirse una bolsa de fichas de distintas denominaciones. Para esto, disponen las fichas desordenadas en una fila, y siguen un juego por turnos, donde en cada turno, el jugador correspondiente saca una ficha desde uno de los extremos de la fila.

Parte 1

Defina inductivamente la ganancia máxima que puede obtener un jugador, asumiendo que el otro jugador juega de forma perfecta, y describa un algoritmo para calcularla.

Idea de solución Se define $G(i,j)$ como la ganancia máxima obtenible, al comenzar a jugar cuando quedan las fichas entre las posiciones i y j. Esta función se define inductivamente como:

$$G(i,j) = \begin{cases} 0 & i > j \\ A[i] & i = j \\ \max\{A[i], A[j]\} & i+1 = j \\ \max\{A[i] + \min\{G(i+2,j), G(i+1,j-1)\}, A[j] + \min\{G(i,j-2), G(i+1,j-1)\}\} & \end{cases}$$

Luego, se puede calcular utilizando programación dinámica:

```
function GananciaMaxima(A[1,...,n])
    G ← Matriz de ceros de n × n
    for ℓ = 0 to n do
        for j = ℓ to n do
            i ← j − ℓ
            if i = j then
                G(i,j) ← A[i]
            else if i + 1 = j then
                G(i,j) ← max{A[i], A[j]}
            else
                G(i,j) ← max{A[i] + min{G(i+2,j), G(i+1,j−1)}, A[j] + min{G(i,j−2), G(i+1,j−1)}}
            end if
        end for
    end for
    return G(1,n)
end function
```

Este algoritmo funciona en tiempo $O(n^2)$, por que se asigna una cantidad cuadrática de posiciones, y cada una se calcula en tiempo constante.

Parte 2

Extienda el resultado anterior para calcular una estrategia óptima para cada jugador, que permita decidir cual ficha tomar en tiempo $O(1)$.

Idea de solución Se busca construir una matriz E donde en cada posición i,j, indique si es conveniente tomar la ficha de la izquierda o de la derecha. Para esto se utiliza un algoritmo similar al anterior, pero donde además de asignar $G(i,j)$, se asigna $E(i,j)$ con el valor i o j según corresponda.

Problema 3

Se busca dividir un string de largo n en $m+1$ substrings, con cortes en las posiciones $\{p_1, \ldots, p_m\}$. Para esto, se dispone de una operación que recibe un string y una posición, y entrega los dos subtrings correspondientes. Esta operación funciona copiando el string recibido como entrada, por lo que su costo es igual al largo del string recibido como entrada.

Describa un algoritmo que determine el costo mínimo de cortar el string en las $m+1$ partes pedidas.

Idea de solución Se define $p_0 = 0$, y $p_{m+1} = n$. Luego, se define $C(i,j)$ como el costo óptimo de cortar el string entre las posiciones p_i, p_j.

Esta función se puede expresar como:

$$C(i,j) = \begin{cases} 0 & i \geq j \\ 0 & i+1 = j \\ p_j - p_i + \min\{C(i,k) + C(k,j) | i < k < j\} & i+1 < j \end{cases}$$

Esto se puede calcular con programación dinámica, siguiendo el mismo orden del problema 2:

function CortarString($\{p_0, \ldots, p_{m+1}\}$)
  $C \leftarrow$ Matriz de ceros de $(m+2) \times (m+2)$
  for $\ell = 2$ **to** n **do**
    for $j = \ell$ **to** n **do**
      $i \leftarrow j - \ell$
      $C(i,j) \leftarrow p_j - p_i + \text{mín}\{C(i,k) + C(k,j) | i < k < j\}$
    end for
  end for
  return $C(0, m+1)$
end function

Este algoritmo funciona en tiempo $O(n^3)$, por que se debe completar una cantidad cuadrática de posiciones, y cada una se calcula en tiempo lineal.

Problema 4

Dado un arreglo de enteros A, decimos que (i, j) es una *inversión*, si $i < j$, pero $A[i] > A[j]$. Desarrolle un algoritmo que permita contar la cantidad de inversiones en un arreglo.

Idea de solución Modificando mergesort. Se utiliza un contador global, que se modifica desde la función merge. Esta función se implementa de la forma usual, pero cuando se elige el elemento desde el lado derecho, se suma al contador la cantidad de elementos restantes al lado izquierdo.