Pontificia Universidad Católica de Chile
Facultad de Ingenieria
Departamento de Ciencias de la Computación
IIC2283 - Diseño y Análisis de Algoritmos

Ayudantía 4

7 de septiembre, 2018

Ayudante: Andrés Ríos

Problema 1

1. La envoltura convexa de un conjunto de puntos X se define como el conjunto convexo más pequeño que contiene a X. La envoltura convexa de X siempre consiste en un poliedro donde los vértices son puntos en X. Demuestre que cualquier algoritmo basado en comparaciones que calcule la envoltura convexa de un conjunto X de n puntos bidimensionales toma $\Omega(n \log(n))$ pasos en el peor caso. Asuma que la envoltura convexa se describe como una lista de vertices en sentido anti-horario.

   Idea de Solución:

   Dada una lista de números $A = [a_1, a_2, ..., a_n]$, generamos los puntos $X = [(a_1, a_1^2), (a_2, a_2^2), ..., (a_n, a_n^2)]$. Es fácil demostrar que todos los puntos en X forman parte de la envoltura convexa de X (la función $f(x) = x^2$ es convexa).

   Sea G un algoritmo basado en comparaciones que retorna la envoltura convexa de un conjunto de puntos bidimensionales. Luego, si tomamos $V' = G(X)$ y definimos $V = [v_1, ..., v_n]$ como la primera coordenada de cada punto en V', tenemos los distintos elementos de $[a_1, ..., a_n]$ ordenados, pero potencialmente rotados. Luego basta recorrer V para encontrar el mínimo - digamos, en el índice i - y luego retornar $[v_i, ..., v_n, v_1, ..., v_{i-1}]$.

   Sea $t_G(n)$ el número de pasos que toma G en determinar la envoltura convexa y $t_H(n)$ el número de pasos que toma el resto del algoritmo. Sabemos que todo algoritmo de ordenación basado en comparaciones toma $\Omega(n \log(n))$ pasos en el peor caso. Por lo tanto, tenemos que

   $$cn \log(n) \leq t_H(n) + t_G(n)$$

   $$cn \log(n) \leq \Theta(n) + t_G(n)$$

   lo que implica que $t_G(n) \in \Omega(n \log(n)))$.

2. Dado un conjunto de puntos X de tamaño n, se busca el par de puntos más cercanos. Demuestre que cualquier algoritmo basado en comparaciones que resuelva este problema toma $\Omega(n \log(n))$ pasos en el peor caso.

   Idea de Solución:

   Asumamos que existe un algoritmo G capaz de encontrar el par de puntos más cercanos en un conjunto de puntos bidimensionales, osea, $G([x_1, ..., x_m]) = (x_i, x_j)$ donde la distancia entre x_i y x_j es mínima y $i \neq j$.

   Podemos definir las funciones

   $$H_1([a_1, a_2, ..., a_n]) = [(a_1, 0), (a_2, 0), ..., (a_n, 0)]$$

   $$H_2(x_i, x_j) = \begin{cases} 1 & x_i = x_j \\ 0 & x_i \neq x_j \end{cases}$$

Sea $A = [a_1, a_2, ..., a_n]$. Tenemos que $H_2(G(H_1(A))) = 1$ si y solo si A tiene elementos repetidos. La ayudantía pasada vimos que todo algoritmo basado en comparaciones que determine si existen elementos repetidos en un arreglo debe realizar $\Omega(n \log(n))$ operaciones en el peor caso.

Por lo tanto,

$$cn\log(n) \leq t_{H_1}(n) + t_G(n) + t_{H_2}(2)$$
$$cn\log(n) \leq \Theta(n) + t_G(n) + \Theta(1)$$

Por lo tanto, tenemos que $t_G \in \Omega(n \log(n))$.

Problema 2

Tienes la oportunidad de invertir en una compañía C. El problema es que el valor de las acciones de C es volátil y cambia mucho de un día para otro. Solo tienes permitido comprar una acción y luego venderla en una fecha posterior. Para compensar esta restricción, tú sabes el valor de la acción en los días futuros (se asume que el valor de la acción es fijo para cada día).

Dado un arreglo $C[1...n]$ que contiene el valor de la acción durante los próximos n días, entregue un algoritmo que compute la máxima utilidad posible al comprar y vender esa única acción.

Idea de Solución:

Se desea encontrar el máximo valor de $C[j] - C[i]$ sobre todo i, j con $j >= i$. Sea $A[2...n]$ tal que $A[i] = C[i] - C[i-1]$. Notar que

$$\begin{aligned}\sum_{t=i}^{t=j} A[t] &= \sum_{t=i}^{t=j} C[t] - C[t-1] \\ &= C[j] - C[j-1] + C[j-1] - C[j-2] + C[j-2] + ... + C[i+1] - C[i] \\ &= C[j] - C[i]\end{aligned}$$

Luego el problema se reduce a encontrar el subarreglo de A que tenga suma máxima.

Dado un índice k en el arreglo A, tenemos 3 opciones: el arreglo de suma máxima está a la izquierda de k, está a la derecha de k, o contiene a k. Sea `MaxSum`(A, i, j) la suma del arreglo de suma máxima entre los índices i y j, y sea `MaxSumAux`(A, i, j, k) la suma del arreglo de suma máxima entre los índices i y j donde $i \leq k \leq j$.

Podemos plantear la siguiente recurrencia para `MaxSum`:

$$MaxSum(A,i,j) = \begin{cases} 0 & i > j \\ max(MaxSum\left(i, \left\lfloor \frac{i+j}{2} \right\rfloor - 1\right), & \\ \quad MaxSum\left(\left\lfloor \frac{i+j}{2} \right\rfloor + 1, j\right), & i \leq j \\ \quad MaxSumAux\left(i, j, \left\lfloor \frac{i+j}{2} \right\rfloor\right)) & \end{cases}$$

Con esto el algoritmo es:

function MaxSum(A, i, j)
  $k := \left\lfloor \frac{i+j}{2} \right\rfloor$
  $left := MaxSum(A, i, k-1)$
  $right := MaxSum(A, k+1, j)$
  $mid := MaxSumAux(A, i, j, k)$
  return $max(left, right, mid)$

```
function MaxSumAux(A, i, j, k)
    left := 0
    sum := 0
    for idx := k − 1 downto i do
        sum := sum + A[idx]
        if sum > left then
            left := sum
    right := 0
    sum := 0
    for idx := k + 1 upto j do
        sum := sum + A[idx]
        if sum > right then
            right := sum
    return left + right + A[k]
```

El algoritmo corre en tiempo $T(n) = 2T(n/2) + n$, lo que es $\Theta(n \log(n))$ por el teorema maestro.

Problema 3

1. Vasya se fue de vacaciones y desea realizar actividades el máximo número de días. Cada día Vasya tiene 3 opciones: descansar, ir al gimnasio (si está abierto) o participar en una competencia (si se realiza una ese día). Para los próximos n días, él sabe si el gimnasio estará abierto o si se realizará una competencia. La única restricción es que Vasya no quiere realizar la misma actividad en dos días seguidos: si fue al gimnasio el día i, entonces no irá al gimnasio el día $i + 1$. Lo mismo aplica para las competencias.

   Dado dos arreglos binarios $g[1...n]$ y $c[1...n]$ donde $g[i]$ indica si el gimnasio estará abierto y $c[i]$ indica si habrá una competencia el día i, entregue un algoritmo que calcule el mínimo número de días que Vasya deberá descansar siguiendo las reglas anteriores.

   Solución:

   Sean $D(i)$, $G(i)$, $C(i)$ funciones que entregan el mínimo número de días que Vasya debe descansar entre los días $1, ..., i$ si el día i se descansa, va al gimnasio o compite, respectivamente. Si en el día i el gimnasio está cerrado entonces se define $G(i) = n$, y si no hay competencia entonces $C(i) = n$. Ahora podemos definir las siguientes recurrencias:

$$D(i) = \begin{cases} 1 & i = 0 \\ min(D(i-1), G(i-1), C(i-1)) + 1 & i > 0 \end{cases}$$

$$G(i) = \begin{cases} (1 - g[i]) \cdot n & i = 0 \\ g[i] \cdot min(D(i-1), C(i-1)) + (1 - g[i]) \cdot n & i > 0 \end{cases}$$

$$C(i) = \begin{cases} (1 - c[i]) \cdot n & i = 0 \\ c[i] \cdot min(D(i-1), G(i-1)) + (1 - c[i]) \cdot n & i > 0 \end{cases}$$

   Luego el resultado final buscado es $min(D(n), G(n), C(n))$.

   Con esto, el algoritmo es:

```
function MinR(n, g[1...n], c[1...n])
    D[1...n] := [0, ..., 0]
    G[1...n] := [0, ..., 0]
    C[1...n] := [0, ..., 0]
    D[1] := 1
    G[1] := (1 − g[1]) · n
```

```
C[1] := (1 − c[1]) · n
for i := 2 upto n do
    D[i] := min(D[i − 1], G[i − 1], C[i − 1]) + 1
    if g[i] = 1 then
        G[i] := min(D[i − 1], C[i − 1])
    else
        G[i] := n
    if c[i] = 1 then
        C[i] := min(D[i − 1], G[i − 1])
    else
        C[i] := n
return min(D[n], G[n], C[n])
```

El algoritmo corre en tiempo $\Theta(n)$.

2. Dada una secuencia $X = \langle x_1, x_2, ..., x_m \rangle$, otra secuencia $Z = \langle z_1, z_2, ..., z_k \rangle$ es una **subsecuencia** de X si existe una secuencia estrictamente creciente de indices $\langle i_1, i_2, ..., i_k \rangle$ tal que $\forall j = 1, 2, ..., k \;\; x_{i_j} = z_j$.

   Dadas dos secuencias $X = \langle x_1, x_2, ..., x_m \rangle$ e $Y = \langle y_1, y_2, ..., y_n \rangle$, entregue un algoritmo que compute el largo de la subsecuencia más larga común a X e Y.

   Solución:

   Sea X_i el prefijo de X de largo i.

   Dado $X = \langle x_1, x_2, ..., x_m \rangle$, $Y = \langle y_1, y_2, ..., y_n \rangle$ y $Z = \langle z_1, z_2, ..., z_k \rangle$, donde Z es una subsecuencia de tamaño máximo (LCS de ahora en adelante) de X e Y, tenemos que

   a) Si $x_m = y_n$, entonces $z_k = x_m = y_n$ y Z_{k-1} es una LCS de X_{m-1} y Y_{n-1}

   b) Si $x_m \neq y_n$, entonces $z_k \neq x_m$ implica que Z es una LCS de X_{m-1} e Y

   c) Si $x_m \neq y_n$, entonces $z_k \neq y_n$ implica que Z es una LCS de X e Y_{n-1}

   Sea $LCS(i, j)$ el largo de la secuencia más larga entre X_i e Y_j. Por la propiedad anterior podemos definir la recurrencia

$$LCS(i,j) = \begin{cases} 0 & i = 0 \vee j = 0 \\ LCS(i-1, j-1) & i, j > 0 \wedge x_i = y_j \\ max(LCS(i, j-1), LCS(i-1, j)) & i, j > 0, x_i \neq y_j \end{cases}$$

   Con esto, el algoritmo es:

```
function LCS(X, Y, m, n)
    c[1...m, 1...n] = [0...0, 0...0
    for i := 1 upto m do
        c[i, 0] = 0
    for j := 1 upto n do
        c[0, j] = 0
    for i := 1 upto n do
        for j := 1 upto m do
            if x_i == y_j then
                c[i, j] := c[i − 1, j − 1] + 1
            else if c[i − 1, j] ≥ c[i, j − 1] then
                c[i, j] = c[i − 1, j]
            else
                c[i, j] = c[i, j − 1]
    return c[m, n]
```

   El algoritmo corre en tiempo $\Theta(|X| \cdot |Y|)$.