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

Ayudantía 11

21 de Noviembre, 2018

Problema 1

Dados $a, b \in \mathbb{N}$ con $b \geq 1$, recuerde que `EXP`(a, b) calcula a^b

Sea `EsPotenciaRango`(l, n) un procedimiento que, dados $l, n \in \mathbb{N}$ tales que $l, n \geq 2$, verifica si existen $a, b \in \{2, ..., n\}$ tales que $l = a^b$. En esta pregunta usted debe desarrollar un algoritmo que implemente `EsPotenciaRango`(l, n) y realice a lo más $c \cdot n$ llamadas a la función `EXP`, donde c es una constante fija. Debe además justificar por qué su algoritmo realiza este número de llamadas.

Importante: El algoritmo que va a desarrollar en esta pregunta sólo puede utilizar las operaciones de comparación $=$, $>$ y $<$, y las funciones $+$ y $-$.

Solución:

Un algoritmo es el siguiente:

```
function EsPotenciaRango(l, n)
    a := 2
    b := n
    while a ≤ n and b ≥ 2 do
        t := Exp(a, b)
        if t == l then
            return true
        else if t < l then
            a := a + 1
        else
            b := b − 1
    return false
```

La idea es la siguiente: tenemos un puntero a una base a y otro puntero a una base b tal que sabemos que las bases en $2, ..., a-1$ y los exponentes en $b+1, ..., n$ no son candidatos válidos para la solución. Inicialmente esto es verdadero: Los rangos $2, ..., 1$ y $n+1, ..., n$ están vacíos, por lo que no estamos descartando ninguna solución.

Luego, si $Exp(a, b)$ es igual a t entonces podemos retornar que si existen a, b tales que $l = a^b$. Si $Exp(a, b)$ es menor a l, entonces sabemos que la base a no es un candidato válido, puesto que a elevado a todos los exponentes candidatos también es menor a l. En este caso podemos aumentar a en 1. Si $Exp(a, b)$ es mayor a l entonces podemos disminuir b en 1 utilizando un razonamiento análogo al anterior.

Tenemos que en cada iteración del `while` se ejecuta una llamada a `Exp` y se aumenta a en 1 o disminuye b en 1. Solo se puede aumentar a y disminuir b $n-1$ veces cada uno. Por lo tanto, no se pueden realizar más de $2(n-1) = 2n-2$ llamadas a `Exp`. Por lo tanto, la cantidad de llamadas a `Exp` está acotada por $2 \cdot n$.

Problema 2

Un ladrón entra a una tienda con una mochila que puede contener k objetos. En la tienda hay n productos distintos donde el costo del i-ésimo producto es a_i, con $1 \leq a_i \leq n$. Hay una cantidad ilimitada de cada producto.

Desarrolle un algoritmo que entregue el número de posibles costos totales que el ladrón puede meter a su mochila, es decir, el algoritmo debe computar la cardinalidad del siguiente conjunto:

$$\{sum(L) \mid L = [b_1, ..., b_k] \wedge \forall j \in \{1, ..., k\} \exists i \;\; b_j = a_i\}$$

Analice el tiempo de ejecución del algoritmo.

Solución:

Sea el polinomio $P(x) = \sum_{i=0}^{n-1} b_i \cdot x^i$, donde $b_i = \begin{cases} 1 & \exists j : a_j = i \\ 0 & \texttt{e.o.c.} \end{cases}$

En otras palabras, el coeficiente b_i de $P(x)$ es no nulo sí y solo sí existe un producto con costo i.

Sea $C(x) = P(x) * P(x) = \sum_{i=0}^{2n-1} c_i \cdot x^i$. Notar que $c_i = \sum_{j+k=i} b_j \cdot b_k$. Esto implica que el i-ésimo coeficiente de $C(x)$ es no nulo sí y solo sí hay un par de coeficientes b_j, b_k tales que ambos son no nulos y $j + k = i$. En otras palabras, c_i es no nulo sí y solo sí hay un par de productos tales que la suma de sus precios es i.

Por lo tanto, si la mochila pudiera contener solo $k = 2$ objetos entonces la respuesta sería la cantidad de coeficientes no nulos de $C(x)$. Es fácil demostrar por inducción que para un k arbitrario, la respuesta es la cantidad de coeficientes no nulos de $(P(x))^k$.

Con esto, el algoritmo es: transformar $P(x)$ a su representación en puntos-valores utilizando al menos nk puntos, elevar cada elemento de esta representación a k, y luego transformar de vuelta a la representación canónica. La respuesta es el número de coeficientes no nulos en esta representación final.

El polinomio final tiene grado acotado por nk, por lo que la cantidad de puntos que hay que tomar para poder aplicar la transformada rápida de Fourier está acotada por $2nk$. Por lo tanto, el costo del algoritmo es $O(2nk \log(2nk) + 2nk \log(k) + 2nk \log(2nk)) = O(nk \log(nk))$.

Problema 3

Una matriz *Toeplitz* es una matriz de A de $n \times n$ tal que $A = (a_{i,j})$ y $a_{i,j} = a_{i-1,j-1}$ para $i = 1, ..., n-1$, $j = 1, ..., n-1$.

1. ¿Es la suma de matrices Toeplitz una matriz Toeplitz? ¿Que hay del producto?

   Solución:

   La suma de matrices Toeplitz sí es una matriz Toeplitz. Sea $A = (a_{i,j}), B = (b_{i,j})$ y $C = A + B = (c_{i,j})$. Tenemos que

   $$c_{i,j} = a_{i,j} + b_{i,j} = a_{i-1,j-1} + b_{i-1,j-1} = c_{i-1,j-1}$$

   Por el otro lado, la multiplicación no es necesariamente una matriz Toeplitz. Tomemos como contraejemplo

   $$A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}, B = \begin{bmatrix} 2 & 3 \\ 4 & 2 \end{bmatrix}, C = A \cdot B = \begin{bmatrix} 10 & 7 \\ 10 & 11 \end{bmatrix}$$

   A y B son matrices Toeplitz, pero su multiplicación no lo es.

2. Describa como representar una matriz Toeplitz de manera de poder sumar dos matrices Toeplitz de $n \times n$ en tiempo $O(n)$.

   Solución:

   Una matriz Toeplitz escrita explícitamente contiene mucha información redundante, ya que sabemos que el valor de cada diagonal es constante. Por lo tanto, nos basta guardar el valor del primer elemento de cada diagonal.

   Dado una matriz Toeplitz $A = (a_{i,j})$, la podemos representar por el vector $c(A)$ definido como

   $$c(A) = [a_{n-1,0}, a_{n-2,0}, ..., a_{1,0}, a_{0,0}, a_{0,1}, ..., a_{0,n-1}]$$

   Si numeramos las diagonales de la matriz de izquierda a derecha, entonces el i-ésimo elemento de $c(A)$ contiene el valor de la diagonal i-ésima.

   Con esta representación es posible sumar dos matrices en tiempo lineal: basta sumar los vectores que las representan elemento por elemento.

3. De un algoritmo $O(n \log(n))$ para multiplicar una matriz Toeplitz de $n \times n$ con un vector de largo n. Use la representación de la parte b.

   Solución:

   Dada la representación $c(A)$ de una matriz Toeplitz A y un vector v, deseamos calcular $y = A \cdot v$. Sea y_i el i-ésimo elemento del vector y, y sea $c(A) = [c_0, ..., c_{2n-2}]$. Es fácil verificar que $a_{i,j} = c_{n-1-i+j}$. Tenemos que

   $$y_i = \sum_{j=0}^{n-1} a_{i,j} \cdot v_j = \sum_{j=0}^{n-1} c_{n-1-i+j} \cdot v_j$$

   Definamos el vector $b = [b_0, ..., b_{n-1}] = [v_{n-1}, v_{n-2}, ..., v_0]$. Tenemos que

   $$y_i = \sum_{j=0}^{n-1} c_{n-1-i+j} \cdot b_{n-1-j}$$

   Notar que los subíndices de los elementos de la sumatoria siempre suman $2n - 2 - i$. De hecho, son todos los pares de elementos tales que sus índices suman $2n - 2 - i$. Por lo tanto, si interpretamos los vectores $c(A)$ y b como las representaciones en forma canónica de dos polinomios $C(x)$ y $B(x)$, tenemos que y_i es el coeficiente que acompaña al término x^{2n-2-i} en la representación canónica de $C(x) \cdot B(x)$. Con esto el problema se reduce a la multiplicación de polinomios, lo que se puede hacer en $O(n \log(n))$ utilizando la transformada rápida de Fourier.

4. De un algoritmo eficiente para mutliplicar dos matrices Toeplitz de $n \times n$. Analice su tiempo de ejecución.

   Solución:

   El algoritmo para calcular $A \cdot B$ es simplemente utilizar el procedimiento anterior para multiplicar cada columna de B con A. Esto toma $O(n \cdot n \log(n)) = O(n^2 \log(n))$. Esto es más eficiente que cualquier otro algoritmo de multiplicación de matrices existente a la fecha, pero solo puede ser aplicado cuando la matriz A es Toeplitz.

Problema 4

Dado un polinomio $A(x)$ con grado acotado por n, se define su i-ésima derivada como

$$A^{(t)}(x) = \begin{cases} A(x) & t = 0 \\ \frac{d}{dx}A^{(t-1)}(x) & 1 \leq t \leq n-1 \\ 0 & t \geq n \end{cases}$$

Desde la representación de coeficientes $(a_0, a_1, ..., a_{n-1})$ de $A(x)$ y un punto x_0, se desea determinar $A^{(t)}(x_0)$ para $t = 0, 1, ..., n-1$.

1. Dado coeficientes b_0, b_1, b_{n-1} tales que

$$A(x) = \sum_{j=0}^{n-1} b_j (x - x_0)^j$$

muestre como calcular $A^{(t)}(x_0)$ para $t = 0, 1, ..., n-1$ en tiempo $O(n)$.

Solución:

Tenemos que

$$\begin{aligned} A^{(0)}(x) &= \sum_{j=0}^{n-1} b_j (x - x_0)^j \\ A^{(0)}(x_0) &= 0! \cdot b_0 \\ A^{(1)}(x) &= \sum_{j=1}^{n-1} j \cdot b_j (x - x_0)^{j-1} \\ A^{(1)}(x_0) &= 1! \cdot b_1 \\ A^{(2)}(x) &= \sum_{j=2}^{n-1} j(j-1) \cdot b_j (x - x_0)^{j-2} \\ A^{(2)}(x_0) &= 2! \cdot b_2 \end{aligned}$$

Se puede demostrar fácilmente por inducción que este patrón continua, y que $A^{(t)}(x_0) = t! \cdot b_t$. Luego se pueden calcular todos los valores de $A^{(t)}(x_0)$ en tiempo lineal con un simple `for` que vaya calculando $t!$ para cada t y lo multiplique con b_t.

2. Explique como encontrar $b_0, b_1, ..., b_{n-1}$ en $O(n \log(n))$, dado $A(x_0 + \omega_n^k)$ para $k = 0, ..., n-1$.

Solución:

Tenemos los valores de $v_k = A(x_0 + \omega_n^k)$ para $k = 0, ..., n-1$. Tenemos que

$$v_k = A(x_0 + \omega_n^k) = \sum_{j=0}^{n-1} b_j (x_0 + \omega_n^k - x_0)^j = \sum_{j=0}^{n-1} b_j (\omega_n^k)^j$$

$$v_k = \sum_{j=0}^{n-1} b_j (\omega_n^k)^j$$

Así, dado los valores de v_k podemos calcular los valores de b_j en tiempo $O(n \log(n))$ utilizando la inversa de la transformada rápida de Fourier.

3. Demuestre que

$$A(x_0 + \omega_n^k) = \sum_{r=0}^{n-1} \left(\frac{\omega_n^{kr}}{r!} \sum_{j=0}^{n-1} f(j) g(r-j) \right)$$

donde $f(j) = a_j \cdot j!$ y

$$g(l) = \begin{cases} x_0^{-l}/(-l)! & -(n-1) \leq l \leq 0 \\ 0 & 1 \leq l \leq n-1 \end{cases}$$

Solución:

Sea $\chi(a) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$.

Tenemos que

$$\begin{aligned} A(x_0 + \omega_n^k) &= \sum_{j=0}^{n-1} a_j (x_0 + \omega_n^k)^j \\ &= \sum_{j=0}^{n-1} a_j \sum_{r=0}^{j} \binom{j}{r} (\omega_n^k)^r x_0^{j-r} \\ &= \sum_{j=0}^{n-1} a_j \sum_{r=0}^{n-1} \binom{j}{r} (\omega_n^k)^r x_0^{j-r} \cdot \chi(j-r) \\ &= \sum_{j=0}^{n-1} a_j \sum_{r=0}^{n-1} \frac{j!}{r!(j-r!)} (\omega_n^k)^r x_0^{j-r} \cdot \chi(j-r) \\ &= \sum_{j=0}^{n-1} \sum_{r=0}^{n-1} \frac{a_j j!}{r!(j-r!)} (\omega_n^k)^r x_0^{j-r} \cdot \chi(j-r) \\ &= \sum_{r=0}^{n-1} \sum_{j=0}^{n-1} \frac{a_j j!}{r!(j-r!)} (\omega_n^k)^r x_0^{j-r} \cdot \chi(j-r) \\ &= \sum_{r=0}^{n-1} \frac{w_n^{kr}}{r!} \sum_{j=0}^{n-1} a_j j! \frac{x_0^{j-r} \cdot \chi(j-r)}{(j-r!)} \\ &= \sum_{r=0}^{n-1} \frac{w_n^{kr}}{r!} \sum_{j=0}^{n-1} f(j) g(r-j) \end{aligned}$$

4. Explique como evaluar $A(x_0 + \omega_n^k)$ para $k = 0, 1, ..., n-1$ en $O(n \log(n))$. Concluya que es posible evaluar todas las derivadas no triviales de $A(x)$ en x_0 en tiempo $O(n \log(n))$.

Solución:

Sea $c_r = \sum_{j=0}^{n-1} f(j) g(r-j)$. Notemos que esto es una convolución, por lo que podemos utilizar FFT para calcular $c_0, c_1, ..., c_{n-1}$ en tiempo $O(n \log(n))$. Luego tenemos que

$$A(x_0 + \omega_n^k) = \sum_{r=0}^{n-1} \frac{c_r}{r!} w_n^{kr}$$

Tenemos los valores de $\frac{c_r}{r!}$ para $r = 0, ..., n-1$, por lo que nuevamente podemos utilizar FFT para calcular los valores de $A(x_0 + \omega_n^k)$ para $k = 0, ..., n-1$ en tiempo $O(n \log(n))$.

Una vez hecho esto podemos utilizar las partes 2 y 1 de esta pregunta para evaluar todas las derivadas no triviales de $A(x)$ en x_0. Dado que solo se realiza una cantidad constante de operaciones $O(n \log(n))$, el proceso completo también es $O(n \log(n))$.