Resuelto por:
Christopher Gómez

Universidad Simón Bolívar
CI-5651 - Diseño de Algoritmos I
Prof. Ricardo Monascal

Tarea 8: Divide y Vencerás/Programación Dinámica

1. Considere un polinomio formado por los números de su carné, donde el i–ésimo número corresponde al coeficiente para x^i.

   Por ejemplo, si su carné es `12-02412`, entonces el polinomio será:

$$\begin{aligned} P(x) &= 1x^0 + 2x^1 + 0x^2 + 2x^3 + 4x^4 + 1x^5 + 2x^6 \\ &= 1 + 2x + 2x^3 + 4x^4 + x^5 + 2x^6 \end{aligned}$$

   Calcule y muestre el resultado de aplicar la DFT (Transformada Discreta de Fourier) al polinomio obtenido, usando las **raíces octavas** de la unidad.

   En este caso, el carné a considerar es `18-10892`, por lo que el polinomio es:

$$\begin{aligned} P(x) &= 1x^0 + 8x^1 + 1x^2 + 0x^3 + 8x^4 + 9x^5 + 2x^6 \\ &= 1 + 8x + x^2 + 8x^4 + 9x^5 + 2x^6 \end{aligned}$$

   Con esto, el vector de coeficientes es $(1, 8, 1, 0, 8, 9, 2, 0)$, ya que debemos completar hasta tener un vector de coeficientes de tamaño 2^k.

   Luego, las raíces octavas de la unidad vendrán dadas por $\omega^k = e^{\frac{\pi k}{4} i}$, con $k \in [0..7]$.

   Corriendo el algoritmo `FFT` con este vector obtenemos el siguiente resultado (comenzando por ω):

$$\begin{aligned} &\mathrm{FFT}\left((1, 8, 1, 0, 8, 9, 2, 0), e^{\frac{\pi}{4} i}\right) \\ &\quad= \Big(29, -7 - i - e^{\frac{\pi}{4} i}, 6 + 17i, -7 + i + e^{\frac{3\pi}{4} i}, \\ &\qquad -5, -7 - e^{\frac{\pi}{2} i} + e^{\frac{\pi}{4} i}, 6 - 17i, -7 + e^{\frac{\pi}{2} i} - e^{\frac{3\pi}{4} i}\Big) \\ &\quad= \left(29, -7 - \frac{\sqrt{2}}{2} - \left(\frac{\sqrt{2}}{2} + 1\right) i, 6 + 17i, -7 + \frac{\sqrt{2}}{2} + \left(1 - \frac{\sqrt{2}}{2}\right) i,\right. \\ &\qquad \left. -5, -7 + \frac{\sqrt{2}}{2} + \left(\frac{\sqrt{2}}{2} - 1\right) i, 6 - 17i, -7 - \frac{\sqrt{2}}{2} + \left(\frac{\sqrt{2}}{2} + 1\right) i\right) \end{aligned}$$

   Lo que corresponde a $\left(P(\omega^0), P(\omega^1), ..., P(\omega^7)\right)$.

2. Considere un número entero positivo X. Definimos la función $\mathtt{decomp}(X)$ como la cantidad de enteros positivos a, b, c y d de tal forma que $ab + cd = X$.

$$\mathtt{decomp}(X) = |\{(a, b, c, d) : a, b, c, d > 0 \wedge ab + cd = X\}|$$

Dado un número N , queremos hallar el máximo valor para $\mathtt{decomp}(X)$ donde $1 \leqslant X \leqslant N$.

Diseñe un algoritmo que permita encontrar la respuesta en $O(N \log N)$.

Nota: Puede suponer que todas las operaciones aritméticas, incluyendo multiplicaciones, divisiones y módulos se hacen en O(1).

Pistas:

- ¿De cuántas formas se puede descomponer N en dos sumandos a y b, tal que $a + b = N$?
- ¿De cuántas formas se puede descomponer N en dos factores a y b, tal que $a \times b = N$?
- ¿Que relación existe entre la cantidad de divisores de un número y su descomposición en factores primos?
- La Criba de Eratóstenes se puede usar para ver si un número es primo. ¿Se podrá modificar para calcular algo más?
- Un cambio de perspectiva pudiera ser de utilidad.

Notemos lo siguiente:

- Para $X < 2$ el valor de $\mathtt{decomp}(X)$ es 0.
- N tiene $N - 1$ descomposiciones en dos sumandos mayores que 0, que vienen dadas por $1 + (N - 1), 2 + (N - 2), ..., (N - 1) + 1$.
- Por otro lado, N tiene tanta descomposiciones en dos factores como divisores, las cuales vendrán dadas por $d_1 \times \left(\frac{N}{d_1}\right), d_2 \times \left(\frac{N}{d_2}\right), ..., d_k \times \left(\frac{N}{d_k}\right)$, donde $d_1, d_2, ..., d_k$ son los divisores de N. Llamemos D_N a la cantidad de divisores de N.
- Para cada forma de descomponer N en dos sumandos, podemos descomponer cada sumando en dos factores de D_i y D_j formas respectivamente.

Teniendo esto en cuenta, podemos expresar el valor de $\mathtt{decomp}(X)$ como:

$$\mathtt{decomp}(X) = \sum_{i=1}^{X-1} D_i D_{X-i}$$

El problema se reduce a hallar el máximo de esta expresión en el rango $1 \leqslant X \leqslant N$.

(Sin terminar ...)

3. Sea $P = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ un conjunto de n puntos.

   Para cualquier subconjunto $C \subseteq P$, definimos la lejanía de C como la multiplicación de la distancia horizontal hasta el origen más grande en C por la distancia vertical hasta el origen más grande en C (nótese que no necesariamente es el mismo punto quien tiene estos máximos).

$$\mathrm{lejania}(C) = \left(\max_{(x,y)\in C} |x|\right) \times \left(\max_{(x,y)\in C} |y|\right)$$

   Queremos realizar una partición de P en subconjuntos $C_1, C_2, ..., C_m$ de tal forma que

   - $C_1 \cup C_2 \cup ... \cup C_m = P$
   - $C_1 \cap C_2 \cap ... \cap C_m = \emptyset$

   La cantidad m de subconjuntos que forma la partición es libre, entre 1 y n. La lejanía de la partición es la suma de las lejanías de los conjuntos que lo conforman.

$$\mathrm{lejania}(C_1 \cup C_2 \cup ... \cup C_m) = \mathrm{lejania}(C_1) + \mathrm{lejania}(C_2) + ... + \mathrm{lejania}(C_m)$$

   Diseñe un algoritmo que permita hallar una partición con mínima lejanía en $O(n \log n)$, usando memoria adicional $O(n)$.

   Pistas:

   - ¿Existen puntos en la entrada que son redundantes?
   - ¿Dar un orden a los puntos nos permitiría considerar **subsecuencias** en lugar de **subconjuntos**?
   - La geometría es un área muy útil de las matemáticas.

   F