Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2283 - Diseño y Análisis de Algoritmos
Profesor: Nicolás Van Sint Jan
Ayudante: Dante Pinto

Ayudantía 6

Transformada de Fourier

Problema 1

Dados n arreglos $A_1, A_2, ..., A_n$, queremos saber cuáles posibles resultados podemos obtener sumando un elemento de cada uno y, además, de cuántas maneras podemos obtenerlo. Por ejemplo para los 3 arreglos $a_1 = [4, 2]$, $a_2 = [1, 1, 3]$ y $a_3 = [2, 3]$, podemos obtener 6 de dos maneras, 5 de dos maneras, 10 de una manera, entre otros.

1. Encuentre una manera de resolver este problema como multiplicación de polinomios.
2. De un algoritmo que resuelva el problema.
3. ¿Cuál es la complejidad del algoritmo?
4. ¿Cómo se comporta el problema cuando los valores de los números de los arreglos son muy grandes?

Solución: 1. Para solucionar este problema bastará con escribir los vectores como polinomios, transformando el arreglo $A_i = [a_1^i, ..., a_n^i]$ en el polinomio:

$$p_i = x^{a_1^i} + ... + x^{a_n^i}$$

Luego, podemos multiplicar todos los polinomios transformados y finalmente, a partir del polinomio $p = c_0x^0 + c_1x^1 ... + c_Nx^N$, podemos interpretar que cada factor c_ix^i nos indica que el número i puede obtenerse de c_i maneras.

2.-3. Considerando la suma y multiplicación de números complejos como operación a contar y utilizando FFT, podemos resolver el problema en tiempo $\mathcal{O}(n \cdot m + n \cdot N \cdot log(N))$, con n la cantidad de arreglos, m el largo máximo de los arreglos y N el máximo valor numérico presente en algún arreglo, siguiendo el algoritmo:

```
1 Function Combinations(A_1, ..., A_n)
2     foreach A_i do
3         p_i = array[N]          //arreglo de 0s
4         foreach a_j ∈ A_i do
5             p_i[a_j]+=1
6         f_i = FFT(p_i)
7     F = MUL(f_i)
8     P = FFT^{-1}(F)
9     return P
```

4. Mirando el tiempo del algoritmo, tenemos un factor que depende de los valores numéricos de los arreglos, por lo que dependiendo de los tamaños y valores del problema, puede que sea conveniente utilizar una solución que no se base en polinomios.

Problema 2

Sea $A = [a_0, ..., a_n]$ un arreglo de números naturales, donde $a_i \leq n$ para todo $i \leq n$. Considere el siguiente conjunto:

$$C = \{(i, j, k) | a_i + a_j = a_k \wedge i \neq j \neq k\}$$

Dado un arreglo A, queremos saber el tamaño del conjunto C.

1. Encuentre una manera de resolver este problema como multiplicación de polinomios.
2. De un algoritmo que resuelva el problema en tiempo $\mathcal{O}(n \log n)$.

Solución: 1. Podemos utilizar la misma idea que para el problema anterior, convirtiendo A en un polinomio p, luego multiplicándolo consigo mismo, obteniendo el polinomio P, y finalmente comparando P con p para comprobar qué elementos del arreglo se pueden generar al sumar otros dos y y cuántas veces se puede generar cada uno. El problema que tiene esta solución es que se estarán contando las tuplas de la forma (i, i, k), donde $a[k] = 2 \cdot a[i]$, y también las tuplas de la forma (i, k, i) y (k, i, i), donde $a_[k] = 0$, que, claramente, no cumplen con que $i \neq j \neq k$.

A pesar de lo anterior, es posible saber de antemano cuántas veces ocurrirán estos errores, pues para el primer caso, las tuplas incorrectas aparecerán una vez por cada $a[i]$ tal que $a[k] = 2 \cdot a[i]$, mientras que para el segundo caso se contarán dos tuplas ((i, k, i) y (k, i, i)) por cada 0 en el arreglo original. Una última consideración es que para los valores 0 en el arreglo, la tupla (i, i, i) cae dentro de los 3 casos anteriores, pero solamente es contada una vez en el algoritmo.

Siguiendo lo anterior, el proceso completo consistirá en, primero, calcular el producto de los polinomios p, obteniendo el arreglo de frecuencias P; segundo, recorrer el arreglo elemento por elemento, restando: 1 a cada $P[2i]$ por cada elemento en p con valor i, 2 a cada $P[i] \neq 0$ por cada elemento en p con valor 0 y 2 a cada $P[i] = 0$ por cada elemento en p con valor 0 distinto de si mismo; y, finalmente, sumar las frecuencias de cada valor a_i una vez por cada vez que aparecen en A.

2. Considerando la suma y multiplicación de números complejos como operación a contar y utilizando FFT, podemos resolver el problema en tiempo $\mathcal{O}(n \cdot log(n))$ siguiendo el algoritmo:

```
1  Function Indexes(a_0, ..., a_n)
2      p = array[n]; s = 0
3      foreach a_i do
4          p[a_i]+=1
5      f = FFT(p)
6      F = MUL(f, f)
7      P = FFT^{-1}(F)
8      // restas
9      for i = 2, 4, ..., n do
10         P[i]-=p[i/2]            //(i, i, k)
11     for i = 1, ..., n do
12         P[i]-=2 · p[0]          //(i, k, i), a_k = 0
13     P[0]-=2(p[0] − 1)           //(i, k, i), a_k = a_i = 0, i != k
14     for i = 1, ..., n do
15         s+=P[i] · p[i]
16     return s
```

Problema 3

Sea $\overline{a}$ un vector dado por $\overline{a} = [a_0, a_1, ..., a_n]$. Las rotaciones cíclicas de $\overline{a}$ serán los vectores obtenidos al mover todos los elementos de $\overline{a}$ una posición a la izquierda y mover el primer elemento a la última posición. Por ejemplo, si tenemos $a = [1, 2, 3, 4]$ sus rotaciones serán $[2, 3, 4, 1]$, $[3, 4, 1, 2]$ y $[4, 1, 2, 3]$.

Dados dos vectores $\overline{a} = [a_0, a_1, ..., a_n]$ y $\overline{b} = [b_0, b_1, ..., b_n]$, queremos obtener el producto punto entre $\overline{a}$ y cada una de las rotaciones cíclicas de $\overline{b}$.

1. Encuentre una manera de resolver este problema como multiplicación de polinomios.
2. De un algoritmo que resuelva el problema.

Solución: 1. Para resolver esto, podemos interpretar $\overline{a}$ directamente como el polinomio p_a, donde:

$$p_a = \sum_{i=0}^{n} a_i \cdot x^i$$

Intuitivamente, podemos intentar hacer la misma interpretación para b, sin embargo al realizar la multiplicación de p_a y p_b siguiendo esta idea obtendremos los factores correctos, pero no en el orden correcto, por lo que tendremos que tomar:

$$p_b = \sum_{i=0}^{n} b_i \cdot x^{n-i}$$

Observando lo obtenido ahora al multiplicar p_a y p_b, notaremos que si bien los factores son correctos y están bien ordenados, algunas de las partes estarán incompletas, por lo que deberemos replicar los coeficientes de b, encontrando:

$$p_b = \sum_{i=0}^{2n} b_{i \mod n} \cdot x^{n-i}$$

Finalmente, multiplicando p_a y p_b encontraremos el polinomio P dado por $\overline{P} = [p_0, p_2, ..., p_n]$ y los valores que nos interesarán serán $\overline{P}[n : 2n]$

2. Considerando la suma y multiplicación de números complejos como operación a contar y utilizando FFT, podemos resolver el problema siguiendo el algoritmo:

```
1 Function Rotation(ā, b̄)
2 |   b̄' = [b_n, ..., b_1, b_0, b_n, ..., b_0]
3 |   foreach a_i do
4 |   |   p[a_i]+=1
5 |   f_a = FFT(ā)
6 |   f_b = FFT(b̄')
7 |   F = MUL(f_a, f_b)
8 |   P = FFT^{-1}(F)
9 |   return P[n : 2n]
```