Convoluciones y FFT

Prerrequisitos

- Aritmética modular
- Combinatoria elemental

Resumen de la clase

Teoría:

- FFT
- NTT (FFT en $\mathbb{Z}_p$)

Aplicaciones:

- Interpretación del producto de polinomios
- String matching con wildcards
- Probabilidades

Enfoque de la clase

- Los problemas de FFT / convoluciones en sí son inusuales
- Ej: Petrozavodsk 2018 - Tritwise Mex
- Lo que es más usual son los problemas de polinomios: problemas que se pueden resolver modelándolos con polinomios (extraer el k-ésimo coeficiente de un polinomio, retornar la suma de sus coeficientes, etc)
- Muy frecuentemente aparecen multiplicaciones de polinomios en estos problemas
- Usando FFT podemos calcular el producto de dos polinomios en $\mathcal{O}(n \log n)$ y resolver problemas que a simple vista parecen imposibles

Problema de motivación

Dados 2 arreglos A, B con $|A|, |B| \leq 3 \cdot 10^5$ y $1 \leq A_i, B_i \leq 10^5$, imprime el número de formas de sumar X con un elemento de A y otro de B para X entre 1 y K. Dos formas se consideran distintas si los índices son distintos.

Ejemplo:

$A = [1, 2, 2, 3]$

$B = [2, 3]$

$K = 6$

Respuesta:

$[0, 0, 1, 3, 3, 1]$

Solución Naive

Una solución es iterar sobre todos los pares y sumar 1 a $ans[a[i] + b[j]]$. Qué complejidad tiene esto? (Hint: no entra XD)

Herramienta nueva: Multiplicación de polinomios!

Desviémonos un poco del problema y consideremos este otro problema:

Cómo multiplicamos polinomios rápidamente?

Problema:

Se te entregan los coeficientes de dos polinomios $A(x)$, $B(x)$, ambos de grado N.

Imprime los coeficientes de $C(x) = A(x) \cdot B(x)$. Recordar que $C_k = \sum_{i+j=k} A_i B_j$

Restricciones:

- $|A_i|, |B_i| \leq 10^9$
- $N \leq 3 \cdot 10^5$

Representación de polinomios en computador

Los polinomios se guardan como una secuencia de coeficientes.

Por ejemplo, el polinomio $A(x) = 9 + 9x + 8x^2 + 2x^5 + 4x^8$ se puede representar mediante el arreglo $A = [9, 9, 8, 0, 0, 2, 0, 0, 4]$.

$A[k]$ guarda el coeficiente de x^k en el polinomio A, por lo que se necesita un arreglo con $\deg(A) + 1$ elementos.

(Recordatorio: El grado de un polinomio es el valor de su exponente más alto)

Abuso de notación ⚠️ ⚠️ ⚠️ ⚠️ 😱 😱

- $A(x)[x^k]$ = Coeficiente que acompaña a x^k en el polinomio $A(x)$
- Lo mismo que $A[k]$
- Si trato a $A(x)$ como un arreglo, me estaré refiriendo a su lista de coeficientes.

¿De dónde sale esa ecuación rara?

$$C(x) = \left(\sum_{i=0}^{N} A_i x^i\right) \left(\sum_{j=0}^{N} B_j x^j\right)$$

$$C(x) = \sum_{i=0}^{N} \sum_{j=0}^{N} A_i B_j x^{i+j}$$

$$C(x) = \sum_{k \geq 0} \left(\sum_{i=0}^{N} A_i B_{k-i}\right) x^k$$

$$C(x)[x^k] = \sum_{i=0}^{N} A_i B_{k-i}$$

$$C(x)[x^k] = \sum_{i+j=k} A_i B_j$$

Solución mala (que mejoraremos)

Iteramos sobre i y k, sumamos $A[i]B[k-i]$ a $C[k]$. Complejidad: $\mathcal{O}(N^2)$. Podemos hacerlo mejor?

La FFT nos permite resolver este problema en $\mathcal{O}(N \log N)$!

Forma punto valor

La lista de coeficientes no es la única forma de representar un polinomio.

Recordemos que un polinomio $P(x)$ de grado N está definido **únicamente** por $N + 1$ puntos $(x, P(x))$, por lo que podemos escoger $N + 1$ puntos del polinomio y utilizarlos como nuestra representación. Si en vez de las listas de coeficientes se nos hubiera entregado esto, podríamos resolver el problema más rápido? (Spoiler: Sí, multiplicando punto a punto y en $\mathcal{O}(N)$)

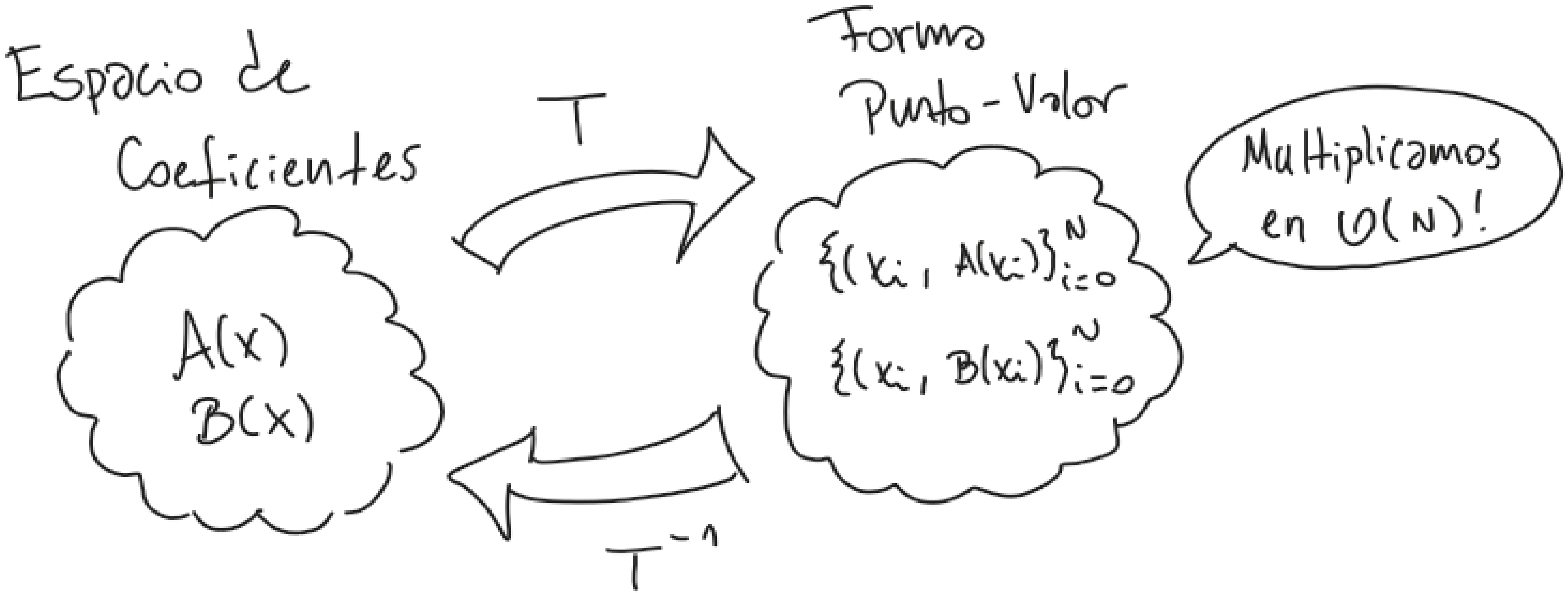
Ejemplo point-value form

Como lista de coeficientes:

- $A(x) = 3 + 2x$
- $B(x) = 2 - 2x$
- $A(x)B(x) = 6 - 2x - 4x^2$

Forma punto-valor (en -1, 0, 1):

- $A = [1, 3, 5]$
- $B = [4, 2, 0]$
- $C = [4, 6, 0]$



Mandar un polinomio de un lado al otro cuesta $O(N^2)$ de forma naive, la FFT hace esto mismo pero en $\mathcal{O}(N \log N)$!

Problema de juguete

Sea P un polinomio de grado N. Cómo calculamos $P(1)$ y $P(-1)$ eficientemente?

Solución subóptima: $2N$ operaciones

- Podemos evaluar ambos polinomios, pero esto no es muy eficiente.

Solución óptima: $N + 2$ operaciones

- Sea s_0 la suma de los coeficientes pares y s_1 la de los coeficientes impares (que se puede calcular en una pasada). $P(1)$ corresponde a $s_0 + s_1$ y $P(-1)$ se puede calcular como $s_0 - s_1$.
- Esto es como decir que, en vez de evaluar un polinomio en -1 y 1, evaluamos **dos** polinomios (exp. pares y exp. impares) de grado $N/2$ en **la mitad** de puntos (de 1 y -1 a sólo 1)!

¿Por qué pasa esto?

Escribamos P como la suma de sus coeficientes pares e impares:

$$P(x) = P_0(x^2) + xP_1(x^2)$$

Ejemplo:

$$P(x) = 1 + x + 2x^2 + 5x^3$$
$$P_0(x) = 1 + 2x$$
$$P_1(x) = 1 + 5x$$

Esta cuestión grita divide and conquer..........

Recién pudimos evaluar un polinomio en dos puntos utilizando sólo uno.

- ¿Podemos evaluar un polinomio en N puntos utilizando $\frac{N}{2}$?
- ¿Qué puntos usamos?

Definiciones

Una raíz N-ésima de la unidad, es un número complejo ω_N tal que

$$\omega_N^N = 1$$

Por simplicidad, se asumirá que se habla de raíces N-ésimas de la unidad a menos que se explicite lo contrario.

Una raíz N-ésima de la undiad ω, es primitiva si **genera** el resto de raíces N-ésimas; es decir, elevar ω a todos los k entre 0 y $N-1$ me va a dar todas las raíces N-ésimas.\

Equivalentemente, ω es primitiva ssi $\omega^k \neq 1$ para todo k entre 1 y $N-1$.

DFT - Discrete Fourier Transform

Definimos la DFT (Discrete Fourier Transform) del vector de coeficientes de un polinomio P como

$$T(P) = (P(\omega^0), P(\omega^1), \ldots, P(\omega^{N-1}))$$

que será la transformación que ocuparemos para ir desde el espacio de coeficientes hasta la forma punto-valor.

```
T(P) { // Pseudocódigo!!!
  if (deg(P) == 0) return P[0]
  P0, P1 = separar(P)
  calcular T(P0) y T(P1) recursivamente
```

```
  return ???
}
```

Usaremos este resultado: $\omega_N^2 = \omega_{\frac{N}{2}}$

Demostración:

No cabe en esta clase

Mergear las llamadas recursivas - Lema útil

Usaremos este resultado: $\omega_N^2 = \omega_{\frac{N}{2}}$

Demostración:

bromita

Mergear las llamadas recursivas - Lema útil

Usaremos este resultado: $\omega_N^2 = \omega_{\frac{N}{2}}$

Demostración:

$$(\omega_N^2)^{\frac{N}{2}} = \omega_N^N = 1$$

Mergear las llamadas recursivas - Ahora sí

Sabemos que $P(x) = P_e(x^2) + xP_o(x^2)$.
Cómo podemos escribir $T(P)$ en términos de $T(P_0)$ y $T(P_1)$?

$$\begin{aligned} T(P)(k) &= P(\omega_N^k) \\ &= P_0(\omega^{2k}) + \omega^k P_1(\omega^{2k}) \\ &= P_0(\omega_{\frac{N}{2}}^k) + \omega^k P_1(\omega_{\frac{N}{2}}^k) \\ &= T(P_1)(k \bmod \deg(P_0)) + \omega^k T(P_1)(k \bmod \deg(P_1)) \end{aligned}$$

Análisis de complejidad

La complejidad del merge es $\mathcal{O}(N)$ y por teorema maestro, la complejidad del algoritmo es $\mathcal{O}(N \log N)$ (Igual que merge sort).

¿Cómo me devuelvo?

- La DFT es una transformación lineal!
- Cada evaluación es una combinación lineal de coeficientes y potencias de raíces primitivas
- Por lo tanto, podemos escribir $DFT(P)$ como $T \cdot P$ donde T es una matriz

La matriz en cuestión

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ P_{N-1} \end{pmatrix} = \begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{N-1}) \end{pmatrix}$$

Matrices de Vandermonde

Esta matriz pertenece a una familia de matrices conocida como **Matrices de Vandermonde**, que guardan progresiones geométricas en cada fila. Estas matrices son invertibles **si y sólo si** ninguna fila se repite (el si es trivial, el si y sólo si no tanto). La inversa de T está dada por

$$T^{-1} = \frac{1}{N}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(N-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \cdots & \omega^{-(N-1)^2} \end{pmatrix}$$

¡Esto es lo mismo que T pero dividido por N y utilizando los inversos de las raíces!

Facto

Esto significa que para calcular la FFT inversa podemos hacer lo mismo pero utilizando ω^{-1} en vez de ω en cada paso, recordando dividir por el largo del polinomio en cada iteración! (O dividir por N al final, lo que les parezca mejor)

Algunos problemas

- Al usar raíces de la unidad, estamos obligados a usar floats, lo que puede causar errores de precisión.
- Esta implementación es recursiva, pero se puede mejorar implementándola iterativamente (leer cpalgorithms)

NTT - chao floats???????

En algunos problemas de conteo, nos piden imprimir la respuesta módulo algún primo p.

Esto es muy útil ya que podemos calcular

$$C_k = \sum_{i+j=k} A_i \cdot B_j \bmod p$$

evitando completamente el uso de floats!

¿Qué necesitamos realmente para hacer FFT?

Recordemos la fórmula que usamos antes:

$$P(\omega_N^k) = P_0(\omega_{\frac{N}{2}}^k) + \omega^k P_1(\omega_{\frac{N}{2}}^k)$$

Ingredientes:

- Un z tal que $z^N = 1$ (Raíz N-ésima)
- Otro z tal que $z^{\frac{N}{2}} = 1$ para la segunda llamada recursiva (Raíz $\frac{N}{2}$-ésima)
- Otro z tal que $z^{\frac{N}{4}} = 1$ para la tercera (Raíz $\frac{N}{4}$-ésima)
- Otro z tal que $z^{\frac{N}{8}} = 1$ para la tercera (Raíz $\frac{N}{8}$-ésima)
- y así sucesivamente...

El resto son potencias de estos elementos.

¿Qué necesitamos realmente para hacer FFT?

Cuando trabajamos en $\mathbb{C}$, los únicos valores que sirven son las raíces de la unidad complejas, pero cuando trabajamos en $\mathbb{Z}_p$ la cosa cambia: Las raíces de la unidad pueden ser enteros!

Ejemplo:

Una raíz 4-ésima de la unidad mod 7 es 6, ya que $6^4 \bmod 7 = 1296 \bmod 7 = 1$.

Cuidado!

Las raíces primitivas podrían no existir para ciertos módulos: Por ejemplo, no existen raíces 5-ésimas módulo 7.

Una pizca de teoría de números

- Las raíces primitivas N-ésimas módulo p existen sólo si N divide a $p - 1$.
- Para que existan raíces N-ésimas para todas las potencias de 2, $p - 1$ debe ser divisible por 2^k, para algún k que será la raíz 2^k-ésima más grande que queremos.
- Esto significa que p debe tener la forma $2^k d + 1$

el mejor numero q existe frfr no cap ong icl ts so tuff osrs

$$998244353 = 119 \cdot 2^{23} + 1$$

NTT

-Ahora que sabemos que las raíces de la unidad existen, debemos encontrarlas. Recordemos que nos basta encontrar una raíz 2^k-ésima, ya que para el resto de potencias de dos podemos elevar la que tenemos al cuadrado!

- Partamos con una raíz primitiva $p - 1$ésima que podemos encontrar googleando en wolframalpha. Para 998244353 se suele usar el 3 o el 5.

Raíces primitivas módulo whatsapp

Sea $p = d \cdot 2^k + 1$ y sea r una raíz primitiva.

Por Pequeño Teorema de Fermat, se tiene esto.

$$\begin{aligned} r^{p-1} &\equiv 1 \bmod p \\ r^{d \cdot 2^k} &\equiv 1 \bmod p \\ \left(r^d\right)^{2^k} &\equiv 1 \bmod p \end{aligned}$$

y como r es raíz primitiva, r^d también es una raíz primitiva mod 2^k! Si usamos estos valores en lugar de las raíces complejas, obtendremos la convolución de nuestros polinomios con los coeficientes módulo p.

Aplicaciones

Volvamos al problema con el que partimos la clase.

- El problema que nos piden se puede resolver calculando el producto de dos polinomios

Interpretación combinatorial del producto de polinomios

- Sumar caminos en un grafo
- Al pasar por un término, se multiplica por el costo actual: Los exponentes se suman y los coeficientes se multiplican. El costo total se agrega a la respuesta y se suma sobre todos los caminos posibles.
- Ejemplo - Variante de knapsack: Se tienen N objetos con pesos $A_i \leq 100$ y una mochila de capacidad $M \leq 10^5$, cuánto es lo máximo que puedo llevar?

Cross correlation

Convolución del tipo

$$\sum_{j-i=k} A_i B_j$$

Podemos pensarla como

$$\sum_{j+i=k} A_{-i} B_j$$

Como no existen índices negativos, damos vuelta el polinomio: La respuesta nos queda shifteada en n (siguiente diapo)

Cross correlation

$$C_k = \sum_{j-i=k} A_i B_j$$

$$C_k = \left(\sum_{i+j=k} A_{n-i} B_j \right) [x^{n+k}]$$

La implementación de esto es hacer la convolución normal de A al revés y B y luego extraer el coeficiente $(n+k)$-ésimo.

La razón de por qué calculamos esto aparecerá más adelante.

String matching

Sea S una string sobre un alfabeto $\Sigma, y sea P otra string sobre el mismo alfabeto.

En qué posiciones aparece P en S?

Y si permitimos wildcards?

String matching sin wildcards

- Solución preliminar: Resolvemos el problema para cada caracter c por separado. Crearemos strings binarios donde $A_i = 1$ si $S_i = c$ y $A_i = 0$ si no. Lo mismo para B_i y P
- Calculamos un score de "similitud" para cada offset que representa en cuántas posiciones tenemos un match

$$\sum_{j-i=k} A_i \cdot B_j$$

- Si sumamos esto sobre todos los caracteres y obtenemos el largo de P en C_k, significa que tenemos un match en la posición k!
- La compklejidad no es óptima pero esta idea se usa en varios problemas (guiño guiño)

Complejidad: $\mathcal{O}(|\Sigma|(|S| + |P|)\log(|S| + |P|))$

Quitando el Σ (Sorry, you're not a sigma)

Usemos la misma idea de calcular un puntaje de similitud para dos offsets:

- Pensemos en los caracteres como números enteros
- Si dos caracteres son iguales en alguna posición, su diferencia es cero
- Queremos que la suma total en un offset sea cero si y sólo si hay un match
- Puede haber suma cero si hay diferencias negativas. Para arreglar esto, elevamos el término al cuadrado

$$C_k = \sum_{j-i=k} (P_i - S_j)^2$$

- Si C_k es cero, necesariamente todas las diferencias fueron cero y por ende tenemos un match en esa posición!

String matching sin wildcards

Para calcular esta sumatoria, primero la reescribimos

$$C_k = \sum_{j-i=k} P_i^2 - 2P_iT_j + T_j^2$$

y observamos que:

- El primer término se puede precalcular
- El tercero también
- El segundo es una convolución!

Complejidad: $(\mathcal{O}(|S| + |P|) \log(|S| + |P|))$

Ahora con wildcards

Cómo modelamos las wildcards?

- Queremos que matcheen con cualquier cosa
- O sea, que den 0 en cualquier posición
- Les asignaremos el valor 0 y multiplicaremos por ambos strings.
- Queda así

$$\sum P_j S_i (P_j - S_i)^2$$
$$= \sum P_i^3 S_i - 2P_i^2 T_j^2 + P_i T_j^3$$

- Todos estos términos se pueden calcular con las convoluciones de secuencias apropiadas!

Enunciado

Hay un juego donde participan N jugadores que se dividen en dos equipos: rojo y azul. Sabemos que si hay K jugadores en el equipo rojo, el equipo rojo tendrá una probabilidad de P_k de ganar. Si los jugadores se asignan completamente al azar, cuál es la probabilidad de que el equipo rojo gane si hay K jugadores en su equipo, para cada K entre 1 y N?

Restricciones:

$N \leq 2 \cdot 10^5$

Solución - Problema de probabilidades

Para un K fijo la respuesta es

$$\frac{1}{2^K}\sum_{i=0}^{K}\binom{K}{i}P_i = \frac{K!}{2^K}\sum_{i=0}^{K}\frac{P_i}{i!}\cdot\frac{1}{(K-i)!}$$
$$= \frac{K!}{2^K}\sum_{i+j=K}\frac{P_i}{i!}\cdot\frac{1}{j!}$$

donde la última ecuación es el coeficiente K-ésimo de la convolución de dos secuencias:

$A_i = \frac{P_i}{i!}$
$B_j = \frac{1}{j!}$

DP con polinomios??

Link: https://codeforces.com/problemset/problem/300/D

$$dp(i,j) = \sum_{x_1+x_2+x_3+x_4=i-1} dp(x_1, j-1)dp(x_2, j-1)dp(x_3, j-1)dp(x_4, j-1)$$

$$P_j(x) = P_{j-1}(x)^4 \cdot x$$

- Las potencias se pueden calcular con FFT
- Shifteamos el polinomio porque queremos el coeficiente de $i - 1$ en vez de i