DIVIDE Y VENCERÁS CON FFT

Metodología de programación.

Contenido

El Paradigma Divide y Vencerás y Dos Algoritmos Básicos de la FFT

Joseph-A Fourier observó que cualquier función continua f(x) puede expresarse como una suma de funciones seno - sen(- x + -), cada una de ellas adecuadamente amplificada y desfasada. Su objeto era caracterizar la tasa de transferencia de calor en los materiales.

La transformada, nombrada en su honor, es una técnica matemática que se puede utilizar para el análisis, compresión y síntesis de datos en muchos campos

Dejar f = f0, ... ,Fn-1. Sea un vector de n números complejos.

La transformada discreta de Fourier de F es otro vector de n números complejos F = F0, ... ,Fn-1 cada uno dado por:

$$\widehat{f}(k) = \sum_{n=0}^{N-1} f(n) e^{\frac{-2i\pi kn}{N}}$$

Los algoritmos FFT explotan esa estructura empleando paradigma divide y vencerás. Los desarrollos de los últimos 30 años han dado lugar a una gran cantidad de variaciones del algoritmo básico. Además, el desarrollo de computadoras multiprocesador ha estimulado el desarrollo de algoritmos FFT diseñados específicamente para funcionar bien en tales máquinas.

La FFT es un algoritmo utilizado para convertir una señal desde su dominio original, como el dominio del tiempo, al dominio de la frecuencia. Esta transformación es útil para analizar señales, identificar componentes frecuenciales y realizar operaciones como el filtrado de señales.

Ejemplo, una señal, como una onda de sonido, que está compuesta por diferentes frecuencias. La FFT te permite descomponer esa señal en sus componentes frecuenciales individuales. Por ejemplo, podrías analizar una señal de audio y determinar las frecuencias principales presentes, como los tonos de una canción.

El algoritmo FFT aprovecha una propiedad matemática llamada Transformada de Fourier, que permite descomponer una señal en una combinación de senos y cosenos de diferentes frecuencias.

La FFT es ampliamente utilizada en aplicaciones de procesamiento de señales, como en la compresión de audio, el procesamiento de imágenes, el análisis de espectro y muchas otras áreas.

Divide y vencerás con FFT.

En 1805 Gauss creó el primer algoritmo de “divide y vencerás”, un algoritmo que se utilizó para el cálculo de la transformada de Fourier discreta de forma recursiva.

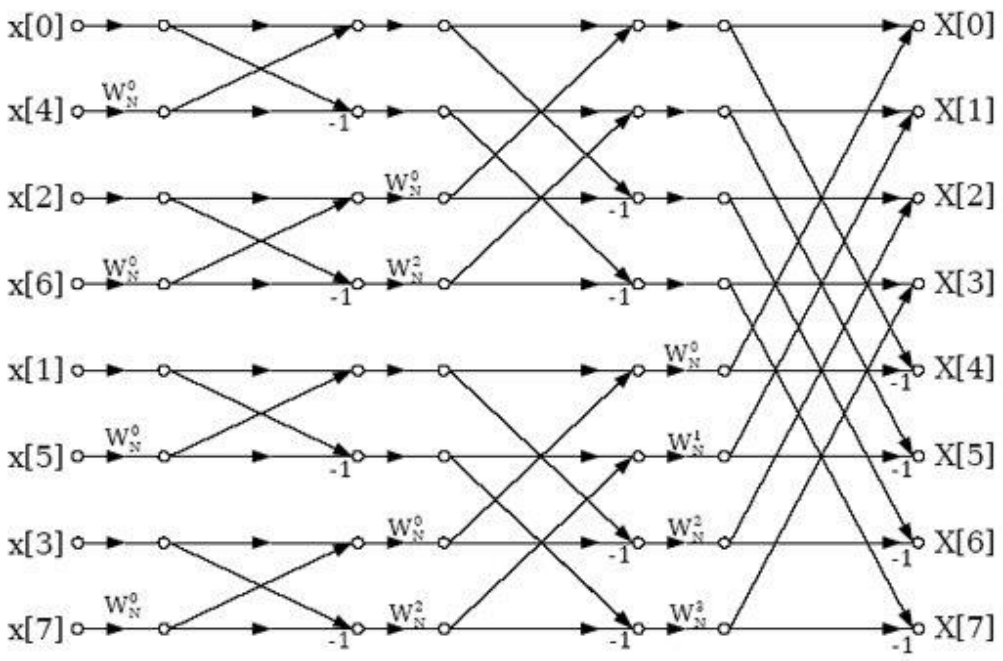
Los tres pasos principales del paradigma dividen y vencerás son:

> Paso 1. Dividir el problema en dos o más subproblemas de menor tamaño.
>
> Paso 2. Resolver cada subproblema de forma recursiva utilizando el mismo algoritmo. Aplicar la condición límite para terminar la recursión cuando los tamaños de los subproblemas son suficientemente pequeños.
>
> Paso 3. Obtener la solución para el problema original combinando las soluciones de los subproblemas.

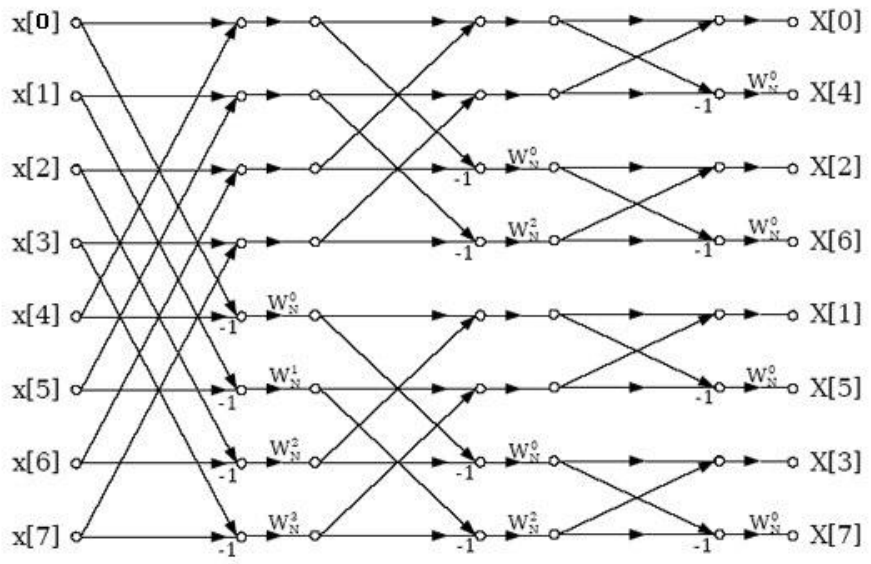
La FFT radix-2 es un algoritmo recursivo que se obtiene al dividir el problema dado (y cada subproblema) en dos subproblemas de la mitad del tamaño.

Dentro de este marco, hay dos variantes de FFT de uso común que difieren en la forma en que se definen los dos subproblemas de tamaño medio. Se les conoce como

- DIT (Decimation in Time) FFT



- DIF (Decimation in frequency) FFT.



Es intuitivamente aparente que una estrategia de divide y vencerás funcionará mejor cuando n es una potencia de dos, ya que la subdivisión de los problemas en problemas sucesivamente más pequeños puede proceder hasta que su tamaño sea uno. Por supuesto, hay muchas circunstancias en las que no es posible que N sea una potencia de dos, por lo que los algoritmos deben modificarse en consecuencia.

Coste aritmético.

Sea T(N) el costo aritmético de calcular la FFT radix-2 DIT de tamaño N, lo cual implica que calcular una transformada de tamaño medio utilizando el mismo algoritmo tiene un costo de T(N/2). Para establecer la ecuación de recurrencia, es necesario relacionar T(N) con T(N/2).

$$X_r = Y_r + \omega_N^r Z_r, \quad r = 0, 1, \ldots, N/2 - 1,$$

$$\begin{aligned} X_{r+\frac{N}{2}} &= \sum_{k=0}^{\frac{N}{2}-1} y_k \omega_{\frac{N}{2}}^{(r+\frac{N}{2})k} + \omega_N^{r+\frac{N}{2}} \sum_{k=0}^{\frac{N}{2}-1} z_k \omega_{\frac{N}{2}}^{(r+\frac{N}{2})k} \\ &= \sum_{k=0}^{\frac{N}{2}-1} y_k \omega_{\frac{N}{2}}^{rk} - \omega_N^r \sum_{k=0}^{\frac{N}{2}-1} z_k \omega_{\frac{N}{2}}^{rk} \\ &= Y_r - \omega_N^r Z_r, \quad r = 0, 1, \ldots, N/2 - 1. \end{aligned}$$

Según las ecuaciones anteriores, se necesitan N sumas complejas y N^2 multiplicaciones complejas para completar la transformada, asumiendo que los factores de giro están precalculados.

Recordemos que una suma compleja requiere dos sumas reales según, y una multiplicación compleja (con resultados intermedios pre-calculados que involucran las partes real e imaginaria de un factor de giro) implica tres multiplicaciones y tres sumas reales.

Por lo tanto, considerando una suma o multiplicación de punto flotante como una operación de punto flotante (flop), se incurre en 2N flops por las N sumas complejas y 3N flops por las N^2 multiplicaciones complejas. En total, se necesitan 5N flops para completar la transformada después de que los dos subproblemas de tamaño medio se resuelven cada uno con un costo de T(N/2). En consecuencia, el costo aritmético T(N) se representa mediante la siguiente ecuación de recurrencia:

$$T(N) = \begin{cases} 2T\left(\frac{N}{2}\right) + 5N & \text{if } N = 2^n \geq 2\,, \\ 0 & \text{if } N = 1. \end{cases}$$

Desarrollando las ecuaciones, obtenemos que el coste es:

$$T(N) = 5N \log_2 N\ .$$

Sabiento esto, podemos obtener la tabla de costes, siguiente:

OPERACION	COSTE
Múltiplicación y suma compleja	O(1)
Reordenamiento de datos	$O(n^2)$*
Mariposa de Cooley-Turkey	O(1)
Total (para una señal de tamaño n)	$O(n^2)$

Ilustración 1 FFT Canonica

OPERACION	COSTE
Múltiplicación y suma compleja	O(1)
Reordenamiento de datos	O(n log n)
Mariposa de Cooley-Turkey	O(1)
Total (para una señal de tamaño n)	O(n log n)

Ilustración 2 FFT Radix-2

Descripción general del algoritmo FFT Radix-2

Tenemos que:

1. Verificar si la longitud de la secuencia es una potencia de 2. Si no lo es, puedes rellenar la secuencia con ceros o truncarla para que tenga una longitud que sea una potencia de 2.
2. Dividir recursivamente la secuencia en dos subsecuencias de igual longitud. La primera subsecuencia contiene los elementos en posiciones pares y la segunda subsecuencia contiene los elementos en posiciones impares.
3. Aplicar la FFT radix-2 en cada subsecuencia de forma recursiva hasta llegar a subsecuencias de longitud 1, que son casos base de la recursión.
4. Combinar los resultados de las subsecuencias para obtener la transformada de Fourier de la secuencia original. Esto se hace utilizando un proceso llamado mezcla (butterfly), que combina los valores de las subsecuencias utilizando operaciones de suma y multiplicación compleja.
5. Repetir el proceso de mezcla para diferentes niveles de la recursión, combinando los resultados de las subsecuencias de longitud cada vez mayor hasta llegar a la secuencia original.

La clave del algoritmo FFT radix-2 es la utilización de raíces de la unidad complejas y la propiedad de simetría de la transformada de Fourier. La recursión y la estrategia de "Divide y vencerás" permiten reducir la complejidad computacional de O(n^2) a O(n log n), lo que hace que la FFT radix-2 sea una de las técnicas más eficientes para calcular la DFT.

```
def FFT(f):
    n = len(f)
    if n==1: return [f[0]]                      # basis case
    F = n*[0]                                   # Initialize results to 0.
    f_even = f[0::2]                            # Divide - even subproblem.
    f_odd  = f[1::2]                            #   "    - odd  subproblem
    F_even = FFT(f_even)                        # recursive call
    F_odd  = FFT(f_odd)                         #   "
    n2 = int(n/2)                               # Prepare to combine results
    for i in range(n2):
        twiddle = exp(-2*pi*1j*i/n)             # These could be precomputed
        oddTerm = F_odd[i] * twiddle            # Odd terms need an adjustment
        F[i]    = F_even[i] + oddTerm           # Compute a new term
        F[i+n2] = F_even[i] - oddTerm           # Compute one more new term
    return F
```

```
FFT(n, [a_0, a_1, ..., a_{n-1}]):
    if n=1: return a_0
    F_even = FFT(n/2, [a_0, a_2, ..., a_{n-2}])
    F_odd  = FFT(n/2, [a_1, a_3, ..., a_{n-1}])
    for k = 0 to n/2 – 1:
        ω^k = e^{2πik/n}
        y^k       = F_even k + ω^k F_odd k
        y^{k+n/2} = F_even k – ω^k F_odd k
    return [y_0, y_1, ..., y_{n-1}]
```

Ejemplos aplicativos de RADIX-2.

Procesamiento de audio en tiempo real: La FFT radix-2 se utiliza en aplicaciones de procesamiento de audio en tiempo real, como ecualizadores de audio digitales. Permite analizar la señal de audio en tiempo real, dividirla en bandas de frecuencia y aplicar ajustes de ecualización específicos en cada banda para mejorar la calidad del sonido.

Análisis de señales biomédicas: La FFT radix-2 se utiliza en el análisis de señales biomédicas, como el electrocardiograma (ECG) y el electroencefalograma (EEG). Permite identificar patrones de frecuencia y detectar anomalías en las señales, lo que es útil en el diagnóstico médico y el monitoreo de la salud.

Procesamiento de imágenes en tiempo real: La FFT radix-2 se utiliza en aplicaciones de procesamiento de imágenes en tiempo real, como la visión por computadora y el reconocimiento de objetos. Permite realizar transformaciones de Fourier rápidas en secuencias de imágenes para detectar características, extraer información relevante y realizar seguimiento de objetos en movimiento.

Bibliografía.

http://dsp-book.narod.ru/FFTBB/0270_PDF_C03.pdf

https://www.ti.com/lit/an/spra152/spra152.pdf

https://www.quora.com/What-is-the-difference-between-decimation-in-time-and-decimation-in-frequency

https://imarrero.webs.ull.es/sctm04/modulo2/15/jsanrosa.pdf