Explicame el algotitmo de transformada rapida de fourier

**ChatGPT dijo:**

ChatGPT

La Transformada Rápida de Fourier (FFT, por sus siglas en inglés) es un algoritmo eficiente para calcular la Transformada Discreta de Fourier (DFT). La DFT toma una señal en el dominio del tiempo y la transforma en una señal en el dominio de la frecuencia. Sin embargo, la DFT tiene una complejidad computacional de O(N2)O(N^2)O(N2), lo que significa que calcularla para una señal de tamaño NNN requiere N2N^2N2 operaciones. El FFT reduce esta complejidad a O(Nlog⁡N)O(N \log N)O(NlogN), lo que lo hace mucho más eficiente, especialmente para señales grandes.

**Concepto básico de la FFT**

La FFT se basa en descomponer la DFT en cálculos más pequeños usando el principio de **divide y vencerás**. La clave es dividir la señal original en dos partes: una parte con los índices pares y otra con los índices impares. Esta descomposición se repite recursivamente hasta que el tamaño de la señal sea 1, donde la DFT es trivial de calcular.

**Pasos del algoritmo FFT**

1. **División de la señal**:
   * Tienes una señal x[n]x[n]x[n] de tamaño NNN, donde NNN es una potencia de 2 (es decir, N=2kN = 2^kN=2k).
   * Se divide la señal en dos subseñales:
     + Una con los elementos en los índices pares: xeven[n]x\_{\text{even}}[n]xeven​[n].
     + Otra con los elementos en los índices impares: xodd[n]x\_{\text{odd}}[n]xodd​[n].
2. **Aplicación recursiva**:
   * Se aplica el FFT de manera recursiva a las dos subseñales, lo que da como resultado las transformadas de Fourier para los elementos pares e impares.
3. **Combinación de los resultados**:
   * Los resultados de los pares e impares se combinan usando la fórmula: X[k]=Xeven[k]+WNkXodd[k]X[k] = X\_{\text{even}}[k] + W\_N^k X\_{\text{odd}}[k]X[k]=Xeven​[k]+WNk​Xodd​[k] X[k+N/2]=Xeven[k]−WNkXodd[k]X[k + N/2] = X\_{\text{even}}[k] - W\_N^k X\_{\text{odd}}[k]X[k+N/2]=Xeven​[k]−WNk​Xodd​[k] donde WNk=e−2πik/NW\_N^k = e^{-2\pi i k / N}WNk​=e−2πik/N es un término llamado "factor de rotación" o "twiddle factor".
4. **Recursión hasta el final**:
   * El proceso continúa dividiendo las subseñales hasta que cada subseñal tenga un solo elemento. En este punto, la DFT es trivial porque la transformada de una señal de un solo valor es el propio valor.

**Ejemplo de una FFT para N=8N = 8N=8**

1. Comenzamos con una señal x[n]x[n]x[n] de tamaño 8. Se divide en los índices pares e impares, obteniendo dos subseñales de tamaño 4.
2. Cada subseñal de tamaño 4 se divide en subseñales de tamaño 2.
3. Finalmente, las subseñales de tamaño 2 se dividen en subseñales de tamaño 1.
4. Se calcula la DFT para estas subseñales de tamaño 1 y luego se combinan usando los factores de rotación.

**Ventajas**

* **Eficiencia**: La FFT reduce el número de operaciones de O(N2)O(N^2)O(N2) a O(Nlog⁡N)O(N \log N)O(NlogN).
* **Aplicaciones**: Se utiliza en análisis de señales, procesamiento de imágenes, compresión de datos y otros campos que requieren transformación de datos entre los dominios del tiempo y de la frecuencia.

Este es el esqueleto básico del algoritmo FFT. Existen diferentes variantes del algoritmo (como la FFT de Cooley-Tukey) que optimizan diferentes aspectos, pero el concepto fundamental es la descomposición recursiva y la combinación eficiente de los resultados.

4o

**Dijiste:**

programame una fft en c++

**ChatGPT dijo:**

ChatGPT

A continuación, te proporciono una implementación básica del algoritmo FFT en C++. Este código utiliza recursión para dividir el problema en partes más pequeñas y combina los resultados de las subseñales.

cpp

Copiar código

#include <iostream>

#include <complex>

#include <vector>

#include <cmath>

const double PI = 3.141592653589793238460;

// Función FFT recursiva

void fft(std::vector<std::complex<double>>& a) {

int n = a.size();

if (n <= 1) return;

// Dividir en pares e impares

std::vector<std::complex<double>> even(n / 2);

std::vector<std::complex<double>> odd(n / 2);

for (int i = 0; i < n / 2; i++) {

even[i] = a[i \* 2];

odd[i] = a[i \* 2 + 1];

}

// Recursión

fft(even);

fft(odd);

// Combinar resultados

for (int k = 0; k < n / 2; k++) {

std::complex<double> t = std::polar(1.0, -2 \* PI \* k / n) \* odd[k];

a[k] = even[k] + t;

a[k + n / 2] = even[k] - t;

}

}

int main() {

// Ejemplo: señal de entrada (tamaño 8, potencia de 2)

std::vector<std::complex<double>> input = {

1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0

};

// Aplicar FFT

fft(input);

// Mostrar resultados

std::cout << "Resultado de la FFT:\n";

for (const auto& val : input) {

std::cout << val << '\n';

}

return 0;

}

**Explicación:**

1. **Estructura de datos**: Utilizamos std::vector<std::complex<double>> para manejar números complejos. La FFT trabaja con números complejos, ya que las señales de frecuencia generalmente tienen una magnitud y una fase (componente real e imaginaria).
2. **División de la señal**: La señal de entrada se divide en dos partes: los índices pares (even) y los impares (odd).
3. **Recursión**: Se aplica la FFT recursivamente a las partes pares e impares.
4. **Combinación**: Una vez calculadas las FFT de los pares e impares, se combinan usando la fórmula con el "factor de rotación" (std::polar genera un número complejo en forma polar).

**Ejecución:**

* El código se puede compilar con cualquier compilador de C++ como g++:

bash

Copiar código

g++ -o fft fft.cpp

./fft

* En este ejemplo, la entrada es una señal de 8 elementos y los resultados de la FFT se imprimen en la salida estándar.

4o

**Dijiste:**

puedes programarlo en C

**ChatGPT dijo:**

ChatGPT

Aquí tienes una implementación de la Transformada Rápida de Fourier (FFT) en C. Este código no usa las bibliotecas de C++ y está hecho de manera más directa utilizando arreglos de números complejos.

c

Copiar código

#include <stdio.h>

#include <math.h>

#include <complex.h>

#define PI 3.14159265358979323846

// Función FFT recursiva

void fft(complex double\* a, int n) {

if (n <= 1) return;

// Dividir en pares e impares

complex double even[n / 2];

complex double odd[n / 2];

for (int i = 0; i < n / 2; i++) {

even[i] = a[i \* 2];

odd[i] = a[i \* 2 + 1];

}

// Recursión

fft(even, n / 2);

fft(odd, n / 2);

// Combinar resultados

for (int k = 0; k < n / 2; k++) {

complex double t = cexp(-2.0 \* I \* PI \* k / n) \* odd[k];

a[k] = even[k] + t;

a[k + n / 2] = even[k] - t;

}

}

int main() {

// Ejemplo: señal de entrada (tamaño 8, potencia de 2)

complex double input[] = {1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0};

int n = sizeof(input) / sizeof(input[0]);

// Aplicar FFT

fft(input, n);

// Mostrar resultados

printf("Resultado de la FFT:\n");

for (int i = 0; i < n; i++) {

printf("%f + %fi\n", creal(input[i]), cimag(input[i]));

}

return 0;

}

**Explicación:**

1. **Estructura de datos**: En C, los números complejos se manejan usando el tipo complex double de la librería estándar <complex.h>. Esto permite realizar operaciones como la suma y multiplicación de números complejos.
2. **División de la señal**: Al igual que en el ejemplo de C++, el array de entrada se divide en dos partes, una para los índices pares (even) y otra para los impares (odd).
3. **Recursión**: Se llama recursivamente a la función fft para las partes pares e impares.
4. **Combinar resultados**: Los resultados de las transformaciones parciales se combinan usando el "factor de rotación" (aquí calculado con cexp, que es la función exponencial compleja).

**Compilación y ejecución:**

Puedes compilar y ejecutar este código en cualquier compilador de C que soporte la librería estándar de números complejos, como gcc:

bash

Copiar código

gcc -o fft fft.c -lm

./fft

El parámetro -lm es necesario para enlazar la librería matemática estándar.

4o

**Dijiste:**

Tradúceme la palabras even y odd

**ChatGPT dijo:**

ChatGPT

Las palabras **even** y **odd** en español se traducen como:

* **even**: par
* **odd**: impar

En el contexto del código, **even** se refiere a los elementos en índices pares, y **odd** a los elementos en índices impares.

4o

**Dijiste:**

muestrame la matriz W en una FFT de 8x8

**ChatGPT dijo:**

ChatGPT

La matriz WWW en la FFT se llama "matriz de coeficientes de Fourier" o "matriz de twiddle factors". En una FFT de tamaño N=8N = 8N=8, la matriz WWW es una matriz compleja de 8×88 \times 88×8 que contiene las potencias de las raíces de la unidad para la transformación.

Para calcular los elementos de la matriz WWW, utilizamos la fórmula:

WNk⋅n=e−2πi⋅k⋅nN=cos⁡(2πknN)−isin⁡(2πknN)W\_{N}^{k \cdot n} = e^{-\frac{2 \pi i \cdot k \cdot n}{N}} = \cos\left(\frac{2 \pi k n}{N}\right) - i \sin\left(\frac{2 \pi k n}{N}\right)WNk⋅n​=e−N2πi⋅k⋅n​=cos(N2πkn​)−isin(N2πkn​)

donde:

* N=8N = 8N=8 (el tamaño de la FFT),
* kkk es el índice de la fila (de 0 a 7),
* nnn es el índice de la columna (de 0 a 7),
* iii es la unidad imaginaria.

Voy a mostrarte los valores de WWW para una FFT de 8×88 \times 88×8, redondeando los números complejos a 2 decimales.

W=[1+0i1+0i1+0i1+0i1+0i1+0i1+0i1+0i1+0i0.71−0.71i0−1i−0.71−0.71i−1+0i−0.71+0.71i0+1i0.71+0.71i1+0i0−1i−1+0i0+1i1+0i0−1i−1+0i0+1i1+0i−0.71−0.71i0+1i0.71−0.71i−1+0i0.71+0.71i0−1i−0.71+0.71i1+0i−1+0i1+0i−1+0i1+0i−1+0i1+0i−1+0i1+0i−0.71+0.71i0−1i0.71+0.71i−1+0i0.71−0.71i0+1i−0.71−0.71i1+0i0+1i−1+0i0−1i1+0i0+1i−1+0i0−1i1+0i0.71+0.71i0+1i−0.71+0.71i−1+0i−0.71−0.71i0−1i0.71−0.71i]W = \begin{bmatrix} 1 + 0i & 1 + 0i & 1 + 0i & 1 + 0i & 1 + 0i & 1 + 0i & 1 + 0i & 1 + 0i \\ 1 + 0i & 0.71 - 0.71i & 0 - 1i & -0.71 - 0.71i & -1 + 0i & -0.71 + 0.71i & 0 + 1i & 0.71 + 0.71i \\ 1 + 0i & 0 - 1i & -1 + 0i & 0 + 1i & 1 + 0i & 0 - 1i & -1 + 0i & 0 + 1i \\ 1 + 0i & -0.71 - 0.71i & 0 + 1i & 0.71 - 0.71i & -1 + 0i & 0.71 + 0.71i & 0 - 1i & -0.71 + 0.71i \\ 1 + 0i & -1 + 0i & 1 + 0i & -1 + 0i & 1 + 0i & -1 + 0i & 1 + 0i & -1 + 0i \\ 1 + 0i & -0.71 + 0.71i & 0 - 1i & 0.71 + 0.71i & -1 + 0i & 0.71 - 0.71i & 0 + 1i & -0.71 - 0.71i \\ 1 + 0i & 0 + 1i & -1 + 0i & 0 - 1i & 1 + 0i & 0 + 1i & -1 + 0i & 0 - 1i \\ 1 + 0i & 0.71 + 0.71i & 0 + 1i & -0.71 + 0.71i & -1 + 0i & -0.71 - 0.71i & 0 - 1i & 0.71 - 0.71i \\ \end{bmatrix}W=​1+0i1+0i1+0i1+0i1+0i1+0i1+0i1+0i​1+0i0.71−0.71i0−1i−0.71−0.71i−1+0i−0.71+0.71i0+1i0.71+0.71i​1+0i0−1i−1+0i0+1i1+0i0−1i−1+0i0+1i​1+0i−0.71−0.71i0+1i0.71−0.71i−1+0i0.71+0.71i0−1i−0.71+0.71i​1+0i−1+0i1+0i−1+0i1+0i−1+0i1+0i−1+0i​1+0i−0.71+0.71i0−1i0.71+0.71i−1+0i0.71−0.71i0+1i−0.71−0.71i​1+0i0+1i−1+0i0−1i1+0i0+1i−1+0i0−1i​1+0i0.71+0.71i0+1i−0.71+0.71i−1+0i−0.71−0.71i0−1i0.71−0.71i​​

Cada elemento Wk,nW\_{k,n}Wk,n​ de esta matriz corresponde a e−2πi⋅k⋅n8e^{-\frac{2 \pi i \cdot k \cdot n}{8}}e−82πi⋅k⋅n​, y cada fila representa una frecuencia específica mientras que cada columna representa una muestra específica de la señal.

4o

**Dijiste:**

bien, gracias, hay valores que se repite, pónle nombre W0 para 1+0i, W1 para 0.71-071i, etc, quiero decir, a los valores iguales ponle un nombre Wx donde x es el el índice. ¿Podrías por favor?

**ChatGPT dijo:**

ChatGPT

¡Claro! Aquí te muestro la matriz WWW de 8×88 \times 88×8 etiquetando los valores repetidos como W0,W1,…W\_0, W\_1, \dotsW0​,W1​,… y así sucesivamente. Para simplificar, los valores únicos se nombran una vez y luego se reutilizan en la matriz.

**Valores únicos:**

* W0=1+0iW\_0 = 1 + 0iW0​=1+0i
* W1=0.71−0.71iW\_1 = 0.71 - 0.71iW1​=0.71−0.71i
* W2=0−1iW\_2 = 0 - 1iW2​=0−1i
* W3=−0.71−0.71iW\_3 = -0.71 - 0.71iW3​=−0.71−0.71i
* W4=−1+0iW\_4 = -1 + 0iW4​=−1+0i
* W5=−0.71+0.71iW\_5 = -0.71 + 0.71iW5​=−0.71+0.71i
* W6=0+1iW\_6 = 0 + 1iW6​=0+1i
* W7=0.71+0.71iW\_7 = 0.71 + 0.71iW7​=0.71+0.71i

**Matriz WWW etiquetada**

W=[W0W0W0W0W0W0W0W0W0W1W2W3W4W5W6W7W0W2W4W6W0W2W4W6W0W3W6W5W4W7W2W1W0W4W0W4W0W4W0W4W0W5W2W7W4W1W6W3W0W6W4W2W0W6W4W2W0W7W6W1W4W3W2W5]W = \begin{bmatrix} W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 \\ W\_0 & W\_1 & W\_2 & W\_3 & W\_4 & W\_5 & W\_6 & W\_7 \\ W\_0 & W\_2 & W\_4 & W\_6 & W\_0 & W\_2 & W\_4 & W\_6 \\ W\_0 & W\_3 & W\_6 & W\_5 & W\_4 & W\_7 & W\_2 & W\_1 \\ W\_0 & W\_4 & W\_0 & W\_4 & W\_0 & W\_4 & W\_0 & W\_4 \\ W\_0 & W\_5 & W\_2 & W\_7 & W\_4 & W\_1 & W\_6 & W\_3 \\ W\_0 & W\_6 & W\_4 & W\_2 & W\_0 & W\_6 & W\_4 & W\_2 \\ W\_0 & W\_7 & W\_6 & W\_1 & W\_4 & W\_3 & W\_2 & W\_5 \\ \end{bmatrix}W=​W0​W0​W0​W0​W0​W0​W0​W0​​W0​W1​W2​W3​W4​W5​W6​W7​​W0​W2​W4​W6​W0​W2​W4​W6​​W0​W3​W6​W5​W4​W7​W2​W1​​W0​W4​W0​W4​W0​W4​W0​W4​​W0​W5​W2​W7​W4​W1​W6​W3​​W0​W6​W4​W2​W0​W6​W4​W2​​W0​W7​W6​W1​W4​W3​W2​W5​​​

Así, cada valor en la matriz se identifica con su respectivo WxW\_xWx​ en lugar de escribir el número complejo completo cada vez.

4o

**Dijiste:**

Si mal no me equivoco el índice correspondería con el valor n donde n es exp(-2\*Pi\*i\*n\*k/N) en la fórmula X[k]=sumatoria desde n=0 a N de X[n]\*exp(-2\*Pi\*i\*n\*k/N)

**ChatGPT dijo:**

ChatGPT

Tienes razón. El índice nnn en la matriz de coeficientes de Fourier WWW de la FFT corresponde al exponente nnn en la expresión e−2πinkNe^{-\frac{2 \pi i n k}{N}}e−N2πink​, donde kkk es el índice de la fila y nnn el índice de la columna. Es decir, el valor WnW\_nWn​ se refiere a la base e−2πinNe^{-\frac{2 \pi i n}{N}}e−N2πin​, lo que hace que cada WnW\_nWn​ sea una raíz NNN-ésima de la unidad, escalada en potencia según el producto k⋅nk \cdot nk⋅n.

Para una matriz FFT de 8×88 \times 88×8 (N=8N = 8N=8), calculamos los valores de Wn=e−2πin8W\_n = e^{-\frac{2 \pi i n}{8}}Wn​=e−82πin​ y los asignamos de la siguiente manera:

Wn=e−2πin8=cos⁡(2πn8)−isin⁡(2πn8)W\_n = e^{-\frac{2 \pi i n}{8}} = \cos\left(\frac{2 \pi n}{8}\right) - i \sin\left(\frac{2 \pi n}{8}\right)Wn​=e−82πin​=cos(82πn​)−isin(82πn​)

**Valores de WnW\_nWn​ para N=8N = 8N=8:**

* W0=1+0iW\_0 = 1 + 0iW0​=1+0i
* W1=0.71−0.71iW\_1 = 0.71 - 0.71iW1​=0.71−0.71i
* W2=0−1iW\_2 = 0 - 1iW2​=0−1i
* W3=−0.71−0.71iW\_3 = -0.71 - 0.71iW3​=−0.71−0.71i
* W4=−1+0iW\_4 = -1 + 0iW4​=−1+0i
* W5=−0.71+0.71iW\_5 = -0.71 + 0.71iW5​=−0.71+0.71i
* W6=0+1iW\_6 = 0 + 1iW6​=0+1i
* W7=0.71+0.71iW\_7 = 0.71 + 0.71iW7​=0.71+0.71i

Ahora, con estos valores, podemos construir la matriz WWW de la FFT de 8×88 \times 88×8, donde cada elemento de la matriz en la posición (k,n)(k, n)(k,n) es Wn⋅kW\_{n \cdot k}Wn⋅k​.

**Matriz WWW de 8×88 \times 88×8 en términos de WnW\_nWn​:**

W=[W0W0W0W0W0W0W0W0W0W1W2W3W4W5W6W7W0W2W4W6W0W2W4W6W0W3W6W5W4W7W2W1W0W4W0W4W0W4W0W4W0W5W2W7W4W1W6W3W0W6W4W2W0W6W4W2W0W7W6W1W4W3W2W5]W = \begin{bmatrix} W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 \\ W\_0 & W\_1 & W\_2 & W\_3 & W\_4 & W\_5 & W\_6 & W\_7 \\ W\_0 & W\_2 & W\_4 & W\_6 & W\_0 & W\_2 & W\_4 & W\_6 \\ W\_0 & W\_3 & W\_6 & W\_5 & W\_4 & W\_7 & W\_2 & W\_1 \\ W\_0 & W\_4 & W\_0 & W\_4 & W\_0 & W\_4 & W\_0 & W\_4 \\ W\_0 & W\_5 & W\_2 & W\_7 & W\_4 & W\_1 & W\_6 & W\_3 \\ W\_0 & W\_6 & W\_4 & W\_2 & W\_0 & W\_6 & W\_4 & W\_2 \\ W\_0 & W\_7 & W\_6 & W\_1 & W\_4 & W\_3 & W\_2 & W\_5 \\ \end{bmatrix}W=​W0​W0​W0​W0​W0​W0​W0​W0​​W0​W1​W2​W3​W4​W5​W6​W7​​W0​W2​W4​W6​W0​W2​W4​W6​​W0​W3​W6​W5​W4​W7​W2​W1​​W0​W4​W0​W4​W0​W4​W0​W4​​W0​W5​W2​W7​W4​W1​W6​W3​​W0​W6​W4​W2​W0​W6​W4​W2​​W0​W7​W6​W1​W4​W3​W2​W5​​​

Cada elemento Wk⋅nW\_{k \cdot n}Wk⋅n​ representa una potencia específica de WnW\_nWn​ utilizada para calcular las transformadas de Fourier en la señal de entrada.

4o

**Dijiste:**

Bien, ahora, usando FFT cómo quedaría esa matriz (teniendo en cuenta que hay que mover filas en la ecuación X[k]=sumatoria desde n=0 a N de X[n]\*exp(-2\*Pi\*i\*n\*k/N)

**ChatGPT dijo:**

ChatGPT

En una implementación de la FFT, la matriz WWW no se calcula explícitamente como en la Transformada Discreta de Fourier (DFT) tradicional. La FFT utiliza un método recursivo que permite reorganizar las operaciones de manera eficiente mediante la "división y conquista", lo que evita el cálculo directo de todos los elementos de WWW. Sin embargo, si quisiéramos estructurar la matriz para reflejar el orden en el que se aplican los coeficientes en la FFT, deberíamos considerar la **permutación en orden de mariposa** (o "butterfly ordering").

**Reordenación en la FFT**

En una FFT de tamaño N=8N = 8N=8:

1. La señal de entrada se reorganiza en **orden de bit-reverso**. Esto significa que cada índice de entrada nnn se convierte en el índice con los bits invertidos.
2. La matriz efectiva que representa la FFT ya no es la misma que la de la DFT, porque la FFT aplica los coeficientes de WWW en un orden reordenado, combinando etapas de tamaño 2, luego 4, luego 8, etc.

**Ejemplo: Matriz WWW de la FFT para N=8N = 8N=8**

Dado el ordenamiento de bit-reverso, las posiciones de los índices serían las siguientes:

* 0→00 \to 00→0
* 1→41 \to 41→4
* 2→22 \to 22→2
* 3→63 \to 63→6
* 4→14 \to 14→1
* 5→55 \to 55→5
* 6→36 \to 36→3
* 7→77 \to 77→7

Por lo tanto, una matriz organizada según la FFT y tomando en cuenta el orden de bit-reverso quedaría así:

FFT Matrix=[W0W0W0W0W0W0W0W0W0W4W2W6W0W4W2W6W0W2W4W6W0W2W4W6W0W6W4W2W0W6W4W2W0W1W2W3W4W5W6W7W0W5W2W7W4W1W6W3W0W3W6W5W4W7W2W1W0W7W6W1W4W3W2W5]\text{FFT Matrix} = \begin{bmatrix} W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 & W\_0 \\ W\_0 & W\_4 & W\_2 & W\_6 & W\_0 & W\_4 & W\_2 & W\_6 \\ W\_0 & W\_2 & W\_4 & W\_6 & W\_0 & W\_2 & W\_4 & W\_6 \\ W\_0 & W\_6 & W\_4 & W\_2 & W\_0 & W\_6 & W\_4 & W\_2 \\ W\_0 & W\_1 & W\_2 & W\_3 & W\_4 & W\_5 & W\_6 & W\_7 \\ W\_0 & W\_5 & W\_2 & W\_7 & W\_4 & W\_1 & W\_6 & W\_3 \\ W\_0 & W\_3 & W\_6 & W\_5 & W\_4 & W\_7 & W\_2 & W\_1 \\ W\_0 & W\_7 & W\_6 & W\_1 & W\_4 & W\_3 & W\_2 & W\_5 \\ \end{bmatrix}FFT Matrix=​W0​W0​W0​W0​W0​W0​W0​W0​​W0​W4​W2​W6​W1​W5​W3​W7​​W0​W2​W4​W4​W2​W2​W6​W6​​W0​W6​W6​W2​W3​W7​W5​W1​​W0​W0​W0​W0​W4​W4​W4​W4​​W0​W4​W2​W6​W5​W1​W7​W3​​W0​W2​W4​W4​W6​W6​W2​W2​​W0​W6​W6​W2​W7​W3​W1​W5​​​

**Explicación:**

* En cada etapa, los coeficientes de Fourier se aplican en grupos, y cada grupo opera con el par de índices invertidos en la siguiente etapa.
* Los valores de WnW\_nWn​ son usados en combinaciones específicas, evitando recalcular la matriz completa.

4o

**Dijiste:**

y cómo quedaría el orden el vector X[n] que multiplica con W

**ChatGPT dijo:**

ChatGPT

En la FFT, el vector de entrada X[n]X[n]X[n] debe reorganizarse en **orden de bit-reverso** antes de comenzar con las multiplicaciones y sumas. Este reordenamiento permite que la FFT divida el problema de la Transformada de Fourier en problemas más pequeños de forma eficiente. Para un tamaño N=8N = 8N=8, el reordenamiento en bit-reverso es el siguiente:

**Reordenamiento en bit-reverso para N=8N = 8N=8**

1. Representamos los índices en binario con 3 bits (ya que log⁡2(8)=3\log\_2(8) = 3log2​(8)=3):
   * 0→0000 \to 0000→000
   * 1→0011 \to 0011→001
   * 2→0102 \to 0102→010
   * 3→0113 \to 0113→011
   * 4→1004 \to 1004→100
   * 5→1015 \to 1015→101
   * 6→1106 \to 1106→110
   * 7→1117 \to 1117→111
2. Invertimos los bits para cada índice:
   * 000→000=0000 \to 000 = 0000→000=0
   * 001→100=4001 \to 100 = 4001→100=4
   * 010→010=2010 \to 010 = 2010→010=2
   * 011→110=6011 \to 110 = 6011→110=6
   * 100→001=1100 \to 001 = 1100→001=1
   * 101→101=5101 \to 101 = 5101→101=5
   * 110→011=3110 \to 011 = 3110→011=3
   * 111→111=7111 \to 111 = 7111→111=7
3. Con este reordenamiento, el vector X[n]X[n]X[n] debe colocarse en el siguiente orden antes de aplicar la FFT:

X[0],X[4],X[2],X[6],X[1],X[5],X[3],X[7]X[0], X[4], X[2], X[6], X[1], X[5], X[3], X[7]X[0],X[4],X[2],X[6],X[1],X[5],X[3],X[7]

**Vector XXX reorganizado para N=8N = 8N=8**

Si originalmente X=[X[0],X[1],X[2],X[3],X[4],X[5],X[6],X[7]]X = [X[0], X[1], X[2], X[3], X[4], X[5], X[6], X[7]]X=[X[0],X[1],X[2],X[3],X[4],X[5],X[6],X[7]], entonces en el orden de bit-reverso se reordena como:

X=[X[0],X[4],X[2],X[6],X[1],X[5],X[3],X[7]]X = [X[0], X[4], X[2], X[6], X[1], X[5], X[3], X[7]]X=[X[0],X[4],X[2],X[6],X[1],X[5],X[3],X[7]]

Este reordenamiento permite que la FFT proceda con sus cálculos de manera eficiente, realizando las combinaciones necesarias en cada etapa de "mariposa" sin recalcular valores innecesarios.

4o

**Dijiste:**

muestrame el algoritmo de la FFT para N=8

**ChatGPT dijo:**

ChatGPT

Para la FFT de tamaño N=8N = 8N=8, usaremos el algoritmo FFT recursivo Cooley-Tukey, que se basa en dividir la señal en partes más pequeñas, aplicando el principio de "divide y vencerás". Este algoritmo reduce la complejidad de O(N2)O(N^2)O(N2) en la DFT directa a O(Nlog⁡N)O(N \log N)O(NlogN) en la FFT.

**Paso a Paso del Algoritmo FFT (Radix-2 Cooley-Tukey) para N=8N = 8N=8:**

1. **Reorganización en orden de bit-reverso**: Reordena el vector de entrada en el orden de bit-reverso, como se describió antes. Por ejemplo, si la entrada es X=[X[0],X[1],X[2],X[3],X[4],X[5],X[6],X[7]]X = [X[0], X[1], X[2], X[3], X[4], X[5], X[6], X[7]]X=[X[0],X[1],X[2],X[3],X[4],X[5],X[6],X[7]], reordénalo en [X[0],X[4],X[2],X[6],X[1],X[5],X[3],X[7]][X[0], X[4], X[2], X[6], X[1], X[5], X[3], X[7]][X[0],X[4],X[2],X[6],X[1],X[5],X[3],X[7]].
2. **Aplicación de mariposas (Butterfly Operations)**: La FFT aplica mariposas en etapas, cada una duplicando el tamaño de las combinaciones hasta alcanzar el tamaño completo de NNN. Cada mariposa usa coeficientes de Fourier Wn=e−2πinNW\_n = e^{-\frac{2 \pi i n}{N}}Wn​=e−N2πin​.
3. **Etapas de la FFT para N=8N = 8N=8**:
   * **Etapa 1**: Aplica mariposas entre pares de elementos contiguos con desplazamiento de 222. Calcula las combinaciones de pares (X[i],X[i+1])(X[i], X[i+1])(X[i],X[i+1]) multiplicando por W0W\_0W0​ y W4W\_4W4​.
   * **Etapa 2**: Aplica mariposas entre grupos de 4 elementos, usando W0W\_0W0​, W2W\_2W2​, W4W\_4W4​, y W6W\_6W6​.
   * **Etapa 3**: Aplica mariposas entre todos los 8 elementos, usando todos los coeficientes WkW\_kWk​ con k=0k = 0k=0 a k=7k = 7k=7.

**Código en C para la FFT de N=8N = 8N=8**

Aquí tienes el código en C para la FFT de tamaño N=8N = 8N=8:

c

Copiar código

#include <stdio.h>

#include <math.h>

#include <complex.h>

#define N 8

#define PI 3.14159265358979323846

void FFT(complex double \*X) {

// Reorganización en orden de bit-reverso

int j = 0;

for (int i = 0; i < N; i++) {

if (i < j) {

complex double temp = X[i];

X[i] = X[j];

X[j] = temp;

}

int m = N / 2;

while (m >= 1 && j >= m) {

j -= m;

m /= 2;

}

j += m;

}

// Aplicación de mariposas

for (int s = 1; s <= log2(N); s++) {

int m = 1 << s; // Tamaño de la subtransformada

complex double wm = cexp(-2.0 \* I \* PI / m); // W\_m = e^(-2\*pi\*i/m)

for (int k = 0; k < N; k += m) {

complex double w = 1.0;

for (int j = 0; j < m / 2; j++) {

complex double t = w \* X[k + j + m / 2];

complex double u = X[k + j];

X[k + j] = u + t;

X[k + j + m / 2] = u - t;

w \*= wm;

}

}

}

}

int main() {

// Entrada de ejemplo

complex double X[N] = {1, 1, 1, 1, 0, 0, 0, 0};

// Llamada a la FFT

FFT(X);

// Resultado de la FFT

printf("Resultado de la FFT:\n");

for (int i = 0; i < N; i++) {

printf("X[%d] = %.2f + %.2fi\n", i, creal(X[i]), cimag(X[i]));

}

return 0;

}

**Explicación del Código:**

1. **Reordenación en bit-reverso**: Se invierten los índices en la primera parte para reorganizar el vector en el orden correcto.
2. **Aplicación de mariposas**: La FFT se calcula en etapas, donde el tamaño de la subtransformada (es decir, mmm) se duplica en cada paso. Los coeficientes Wm=e−2πimW\_m = e^{-\frac{2 \pi i}{m}}Wm​=e−m2πi​ se recalculan en cada paso.
3. **Salida**: La salida del programa imprimirá los valores de la FFT para los coeficientes de frecuencia X[k]X[k]X[k] para k=0k = 0k=0 a 777.

Este código te da el resultado de la FFT para una señal de entrada XXX de tamaño N=8N = 8N=8 usando el algoritmo de Cooley-Tukey en C.

4o

**Dijiste:**

podrías explicarme cómo se hacen las mariposas?

**ChatGPT dijo:**

ChatGPT

Claro, las "mariposas" en la FFT son operaciones que combinan pares de valores con el objetivo de descomponer la señal en subproblemas más pequeños, logrando así una mejora en la eficiencia del cálculo de la Transformada de Fourier.

**¿Qué es una operación de mariposa (butterfly)?**

Una operación de mariposa toma dos valores y los combina aplicando una suma y una resta, multiplicando uno de los valores por un coeficiente de fase o factor de rotación WWW. Esta operación debe su nombre a la forma visual del diagrama que muestra cómo se combinan los valores.

La mariposa usa los coeficientes Wk=e−2πikNW\_k = e^{-\frac{2 \pi i k}{N}}Wk​=e−N2πik​ para cada etapa de la FFT, donde iii es la unidad imaginaria, kkk depende de la etapa y la posición en la señal, y NNN es el tamaño total de la señal.

**Operación de Mariposa: Fórmulas**

En una mariposa, tenemos dos valores de entrada aaa y bbb, y el coeficiente de fase WkW\_kWk​:

1. El primer valor de salida es:

a′=a+b⋅Wka' = a + b \cdot W\_ka′=a+b⋅Wk​

1. El segundo valor de salida es:

b′=a−b⋅Wkb' = a - b \cdot W\_kb′=a−b⋅Wk​

Esto significa que el valor de entrada aaa se "combina" con bbb multiplicado por el factor WkW\_kWk​, y genera dos nuevos valores a′a'a′ y b′b'b′ para la siguiente etapa de la FFT.

**Ejemplo de Mariposa en la FFT para N=8N = 8N=8**

Supongamos que tenemos una señal de entrada de tamaño N=8N = 8N=8: X[0],X[1],…,X[7]X[0], X[1], \ldots, X[7]X[0],X[1],…,X[7], y estamos en la primera etapa de la FFT. Las mariposas combinan pares de valores de entrada en diferentes niveles.

1. **Primera Etapa** (pares de 1):
   * Combina X[0]X[0]X[0] con X[1]X[1]X[1] usando W0=1W\_0 = 1W0​=1
   * Combina X[2]X[2]X[2] con X[3]X[3]X[3] usando W0=1W\_0 = 1W0​=1
   * Combina X[4]X[4]X[4] con X[5]X[5]X[5] usando W0=1W\_0 = 1W0​=1
   * Combina X[6]X[6]X[6] con X[7]X[7]X[7] usando W0=1W\_0 = 1W0​=1
2. **Segunda Etapa** (pares de 2):
   * Combina el resultado de (X[0],X[1])(X[0], X[1])(X[0],X[1]) con el resultado de (X[2],X[3])(X[2], X[3])(X[2],X[3]) usando W0=1W\_0 = 1W0​=1 y W2=e−2πi8⋅2=0.707−0.707iW\_2 = e^{-\frac{2 \pi i}{8} \cdot 2} = 0.707 - 0.707iW2​=e−82πi​⋅2=0.707−0.707i
   * Combina el resultado de (X[4],X[5])(X[4], X[5])(X[4],X[5]) con el resultado de (X[6],X[7])(X[6], X[7])(X[6],X[7]) usando W0=1W\_0 = 1W0​=1 y W2=0.707−0.707iW\_2 = 0.707 - 0.707iW2​=0.707−0.707i
3. **Tercera Etapa** (pares de 4):
   * Combina el resultado de la segunda etapa usando todos los valores de WkW\_kWk​: W0=1W\_0 = 1W0​=1, W1=e−2πi8=0.707−0.707iW\_1 = e^{-\frac{2 \pi i}{8}} = 0.707 - 0.707iW1​=e−82πi​=0.707−0.707i, W2=0.707−0.707iW\_2 = 0.707 - 0.707iW2​=0.707−0.707i, W3=e−2πi⋅38=−0.707−0.707iW\_3 = e^{-\frac{2 \pi i \cdot 3}{8}} = -0.707 - 0.707iW3​=e−82πi⋅3​=−0.707−0.707i

**Ejemplo Numérico Simplificado**

Si tenemos una entrada simple X=[1,2,3,4,0,0,0,0]X = [1, 2, 3, 4, 0, 0, 0, 0]X=[1,2,3,4,0,0,0,0]:

1. **Primera Etapa**:
   * Mariposa entre X[0]X[0]X[0] y X[1]X[1]X[1]: X[0]′=1+2=3,X[1]′=1−2=−1X[0]' = 1 + 2 = 3, \quad X[1]' = 1 - 2 = -1X[0]′=1+2=3,X[1]′=1−2=−1
   * Mariposa entre X[2]X[2]X[2] y X[3]X[3]X[3]: X[2]′=3+4=7,X[3]′=3−4=−1X[2]' = 3 + 4 = 7, \quad X[3]' = 3 - 4 = -1X[2]′=3+4=7,X[3]′=3−4=−1
   * Mariposa entre X[4]X[4]X[4] y X[5]X[5]X[5]: X[4]′=0+0=0,X[5]′=0−0=0X[4]' = 0 + 0 = 0, \quad X[5]' = 0 - 0 = 0X[4]′=0+0=0,X[5]′=0−0=0
   * Mariposa entre X[6]X[6]X[6] y X[7]X[7]X[7]: X[6]′=0+0=0,X[7]′=0−0=0X[6]' = 0 + 0 = 0, \quad X[7]' = 0 - 0 = 0X[6]′=0+0=0,X[7]′=0−0=0
2. **Segunda Etapa**: Combina X[0]′X[0]'X[0]′ con X[2]′X[2]'X[2]′, y así sucesivamente con los valores obtenidos de la primera etapa, usando los coeficientes correspondientes.

Las operaciones de mariposa continúan hasta que todos los valores se hayan combinado en la última etapa. La FFT utiliza estas mariposas para aprovechar la estructura simétrica de la transformada de Fourier y reducir el número de operaciones necesarias.