Algoritmos y Programación II
TP1: Recursividad

Bourbon, Rodrigo
Carreño Romano, Carlos Germán
Sampayo, Sebastián Lucas

Primer Cuatrimestre de 2015

Índice

1. Objetivos

Ejercitar técnicas de diseño, análisis, e implementación de algoritmos recursivos.

2. Introducción

Explicar un poco que es la FT, la DFT y la FFT.

3. Standard de estilo

Adoptamos la convención de estilo de código de Google para C++, salvando las siguientes excepciones:

- Streams: utilizamos flujos de entrada y salida
- Sobrecarga de operadores

`https://google-styleguide.googlecode.com/svn/trunk/cppguide.html#Naming`

4. Diseño del programa

Explicar a grandes rasgos como funciona el programa, diagrama en bloques. -¿Leer de la entrada a vector, rellenar con ceros, transformar, imprimir vector.

5. Opciones del programa

El programa se ejecuta en línea de comandos, y las opciones que admite (sin importar el orden de aparición) son las siguientes:

nombre largo (*nombre corto*): *descripción*

- `--input` (`-i`):
- `--output` (`-o`):
- `--method` (`-m`):

6. Métodos de la Transformada

como fue implementado dft y fft, funciones genéricas, máscaras, complejidad temporal, espacial, etc.

6.1. FFT

6.1.1. Complejidad Temporal

Para estudiar el costo temporal de esta implementación —$T(N)$— se analizó cada línea de código de la función *calculate_fft_generic()*.

Al principio, todas las sentencias son de orden constante hasta que aparece el primer ciclo:

Las únicas expresiones que ofrecen cierta duda de que su coste sea constante son las últimas —constructores de N/2 elementos. Sin embargo, al ver la implementación de dicho constructor no quedan dudas, ya que solo consiste en una comparación, una asignación, y una llamada a *new*:

Luego se tiene un ciclo de N/2 iteraciones cuyas operaciones en cada caso son de orden constante, con lo cual el orden de este ciclo es $\mathcal{O}(N/2)$.

A continuación encontramos las llamadas recursivas. Dado que el tamaño de la entrada se reduce a la mitad, tenemos 2 llamadas de coste $T(N/2)$.

Finalmente, se tiene un ciclo de N iteraciones cuyas operaciones en cada caso son de orden constante, produciendo un coste de $\mathcal{O}(N)$.

De esta forma, agrupando estos resultados parciales, se puede escribir la ecuación de recurrencia para este algoritmo:

$$T(N) = \mathcal{O}(1) + \mathcal{O}(N/2) + 2T(N/2) + \mathcal{O}(N)$$
$$T(N) = 1 + N + 2T(N/2)$$

$$\boxed{T(N) = 2T(N/2) + N}$$

Como se puede ver, es posible aplicar el teorema maestro, definiendo:

$$a = 2 \geq 1$$
$$b = 2 > 1$$
$$f(N) = N$$

Utilizando el segundo caso del teorema:

$$\exists\, k \geq 0 \quad / \quad N \,\epsilon\, \Theta(N^{\log_b(a)} \log^k(N))$$
$$\Rightarrow T(N) \,\epsilon\, \Theta(N^{\log_b(a)} \log^{k+1}(N))$$

Es fácil ver que con $k = 0$ dicha condición se cumple, por lo tanto el resultado final es:

$$\boxed{T(N) \,\epsilon\, \Theta(N \log N)}$$

Este resultado es coherente, ya que el algoritmo utiliza la técnica de "divide y vencerás" la recurrencia es análoga al caso del conocido *MergeSort*.

7. Estructura de archivos

8. Compilación

Como se compila

9. Casos de prueba

los q aparecen en la especificación del tp, mostrar capturas de pantalla de la consola ejecutando todo

10. Código

11. Enunciado