# Algorithmie

Divide and Conquer

BOULCH Alexandre

# Plan de la séance

Divide and Conquer

Quicksort

Tri fusion

Fast Fourier Transform

TP

# Divide and Conquer

*Diviser pour régner*

**Idée**
Diviser un problème en sous-problèmes plus petits, plus faciles à résoudre.

# Plan de la séance

Divide and Conquer

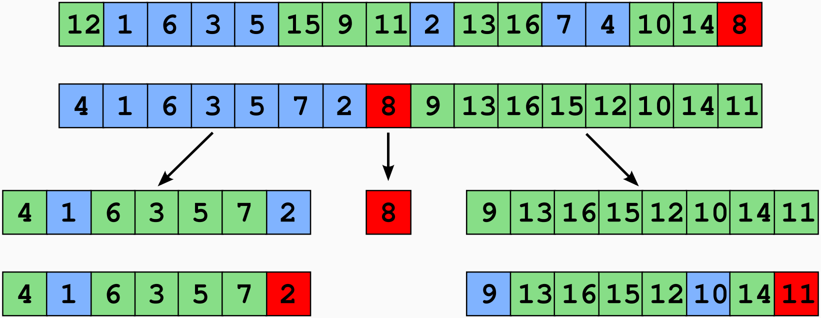
Quicksort

Tri fusion

Fast Fourier Transform

TP

# Principe - Rappel



# QuickSort : complexité

Le parcours du tableau implique $N+1$ comparaison. Puis on réitère l'opération sur chaque moitié de tableau. Le coût est : ($i$ est position du pivot)

$$C_{moy}(N) = N + 1 + Moy_i(C_{moy}(i-1) + C_{moy}(N-i))$$

# QuickSort : complexité

En moyenne le pivot peut se retrouver n'importe où de manière équiprobable : $\frac{1}{N}$

$$Moy_i(C_{moy}(i-1) + C_{N-i}) = \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + C_{moy}(N-p)$$

# QuickSort : complexité

En moyenne le pivot peut se retrouver n'importe où de manière équiprobable : $\frac{1}{N}$

$$Moy_i(C_{moy}(i-1) + C_{N-i}) = \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + C_{moy}(N-p)$$

et donc :

$$C_{moy}(N) = N + 1 + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + C_{moy}(N-p)$$

# QuickSort : complexité

$$C_{moy}(N) = N + 1 + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(N-p)$$

# QuickSort : complexité

$$C_{moy}(N) = N + 1 + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(N-p)$$

un changement de variable dans la second somme donne :

$$C_{moy}(N) = N + 1 + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + \frac{1}{N}\sum_{q=1}^{N} C_{moy}(q-1)$$

# QuickSort : complexité

$$C_{moy}(N) = N + 1 + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(N-p)$$

un changement de variable dans la second somme donne :

$$C_{moy}(N) = N + 1 + \frac{1}{N}\sum_{p=1}^{N} C_{moy}(p-1) + \frac{1}{N}\sum_{q=1}^{N} C_{moy}(q-1)$$

et ainsi :

$$C_{moy}(N) = N + 1 + \frac{2}{N}\sum_{p=1}^{N} C_{moy}(p-1)$$

On multiplie par $N$ :

$$NC_{moy}(N) = N(N+1) + 2\sum_{p=1}^{N} C_{moy}(p-1)$$

et pour $N-1$ :

$$(N-1)C_{moy}(N-1) = N(N-1) + 2\sum_{p=1}^{N-1} C_{moy}(p-1)$$

On multiplie par $N$ :

$$NC_{moy}(N) = N(N+1) + 2\sum_{p=1}^{N} C_{moy}(p-1)$$

et pour $N-1$ :

$$(N-1)C_{moy}(N-1) = N(N-1) + 2\sum_{p=1}^{N-1} C_{moy}(p-1)$$

puis on soustrait :

$$NC_{moy}(N) = 2N + (N+1)C_{moy}(N-1)$$

$$NC_{moy}(N) = 2N + (N+1)C_{moy}(N-1)$$

On divise par $N(N+1)$ :

$$\frac{C_{moy}(N)}{N+1} = \frac{2}{N+1} + \frac{C_{moy}(N-1)}{N}$$

Puis par récurrence :

$$\frac{C_{moy}(N)}{N+1} = \frac{C_{moy}(1)}{2} + \sum_{k=3}^{N+1} \frac{2}{k}$$

Comme :

$$\sum_{k=1}^{N} \frac{1}{k} \sim_{N \to \infty} log(N)$$

il vient :

$$C_{moy}(N) = O(N \log(N))$$

# Plan de la séance

Divide and Conquer

Quicksort

Tri fusion

Fast Fourier Transform

TP

# Tri fusion

Le principe du tri fusion est proche du quicksort. L'idée est de couper un tableau en deux, de trier chaque moitié du tableau, puis de remplir un nouveau tableau en récupérant le bon ordre.

# Tri fusion

1 7 3 9 5 10 4 6 8 2

| | |
|---|---|
| 1 7 3 9 5 | 10 4 6 8 2 |

| | |
|---|---|
| 1 3 5 7 9 | 2 4 6 8 10 |

1 2 3 4 5 6 7 8 9 10

# Calcul de la complexité

Le parcours du tableau implique $N-1$ comparaison. Donc :

$$C_N = (N-1) + C_i + C_{N-i-1}$$

Si on suppose que $i \simeq \frac{N}{2}$ en moyenne :

$$C_N = N + 2 * C_{\frac{N}{2}}$$

Au rang suivant :

$$C_N = 2N + 4 * C_{\frac{N}{4}}$$

Puis :

$$C_N = kN + 2^k * C_{\frac{N}{2^k}}$$

Au final :

$$C_n = N \log N + NC_1 = O(N \log N)$$

(Preuve par récurrence,avec $N \log_2 N$)

# Tri fusion

Le tri fusion est un $O(N \log(N))$ dans tous les cas. Cependant il est en moyenne plus lent que Quicksort, c'est pourquoi ce dernier est le plus utilisé.

# Plan de la séance

Divide and Conquer

Quicksort

Tri fusion

Fast Fourier Transform

TP

# Discrete Fourier Transform

La Transformée de Fourier discrète est un algorithme important en traitement du signal et des images.

# DFT : construction

Soit l'espace complexe $\mathbb{C}^N$ et la forme hermitienne :

$$\langle f, g \rangle = \sum_{j=0}^{N-1} f[k]\overline{g[k]}.$$

La famille des vecteur $e_k$ :

$$e_k = \frac{1}{\sqrt{N}} \begin{pmatrix} e^{\frac{2i\pi}{N}0\cdot k} & e^{\frac{2i\pi}{N}1\cdot k} & \dots & e^{\frac{2i\pi}{N}(N-1)\cdot k} \end{pmatrix}$$

pour $k = 0, \dots, N-1$ est une famille libre orthonormale (donc une base).

# DFT : construction 2

Soit $f$ un tableau de $N$ nombres complexes. Comme $(e_0, e_1, \dots, e_{N-1})$ est une base :

$$f = \sum_{j=0}^{N-1} \langle f, e_j \rangle \, e_j$$

Les coordonnées de $f$ dans la nouvelle base sont celles de la *DFT* de $f$.

**Expression**

La transformée de Fourier discrète transforme un tableau $f$ de $N$ nombres complexes en un tableau $DFT(f)$ de même taille par l'opération suivante :

$$DFT(f)[k] = \langle f, e_k \rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} f[j] e^{-\frac{2i\pi}{N} jk}.$$

# DFT : transformée inverse

Soit $f$ un tableau de $N$ nombres complexes. Comme $(e_0, e_1, \ldots, e_{N-1})$ est une base :

$$f = \sum_{j=0}^{N-1} \langle f, e_j \rangle \, e_j$$

ou encore :

$$f = \sum_{j=0}^{N-1} DFT(f)[j] \, e_j$$

# DFT : transformée inverse

$$f = \sum_{j=0}^{N-1} DFT(f)[j]\, e_j$$

$$f[k] = \left( \sum_{j=0}^{N-1} DFT(f)[j]\, e_j \right) [k]$$

$$f[k] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} DFT(f)[j] e^{+\frac{2i\pi}{N} jk}$$

Notant $IDFT$ la transformée inverse $\left(IDFT \circ DFT = Id\right)$ :

$$f[k] = IDFT(DFT(f))[k] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} DFT(f)[j] e^{+\frac{2i\pi}{N} jk}$$

# DFT

Pour résumer, $f$ un tableau de $N$ nombres complexes :

**Discrete Fourier Transform**

$$DFT(f)[k] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} f[j] e^{-\frac{2i\pi}{N} jk}.$$

**Inverse Discrete Fourier Transform**

$$IDFT(g)[k] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} g[j] e^{+\frac{2i\pi}{N} jk}.$$

# DFT : complexité a priori

$$DFT(f)[k] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} f[j] e^{-\frac{2i\pi}{N}jk}.$$

Calculer un terme : $O(N)$.

Calculer tous les termes : $O(N^2)$

# Fast Fourier Transform

La FFT est un algorithme introduit par Cooley and Tukey en 1965.
Elle permet de calculer la DFT en temps $N \log(N)$.

Elle utilise une approche divise pour régner.

# FFT : algorithme

On sépare la somme dans la DFT en indices pairs et impairs :

$$\sqrt{N} DFT(f)[k] = \sum_{j=0}^{N/2-1} f[2j] e^{-\frac{2i\pi}{N}(2j)k} + \sum_{j=0}^{N/2-1} f[2j+1] e^{-\frac{2i\pi}{N}(2j+1)k},$$

dont on déduit facilement

$$\sqrt{N} DFT(f)[k] = \sum_{j=0}^{N/2-1} f[2j] e^{-\frac{2i\pi}{N/2}jk} + e^{-\frac{2i\pi}{N}k} \sum_{j=0}^{N/2-1} f[2j+1] e^{-\frac{2i\pi}{N/2}jk}.$$

# FFT : algorithme

Il vient :

$$\sqrt{N}DFT(f)[k] = \sqrt{\frac{N}{2}}\left(DFT(f_{pair})[k] + e^{-\frac{2i\pi}{N}k}DFT(f_{impair})[k]\right) \tag{1}$$

où $f_{pair}$ est le sous-tableau des indices pairs de $f$ et $f_{impair}$ est celui des indices impairs.

# FFT : algorithme

Le problème est alors de :

1. Calculer la DFT des indices pairs de $f$.
2. Calculer la DFT des indices impairs de $f$.
3. Combiner en $O(N)$ les deux suivant la formule ci-dessus.

# Complexité

Comme pour le tri fusion :

- calcul sur les tableau de taille $N/2$
- relation de récurrence $C(N) \approx N + 2 * C(\frac{N}{2})$
- complexité en $O(N\log(N))$

# Implémentation

$$\sqrt{N} DFT(f)[k] = \sqrt{\frac{N}{2}} \left( DFT(f_{0:2:N-2})[k] + e^{-\frac{2i\pi}{N}k} DFT(f_{1:2:N-1})[k] \right)$$

Relation de récurrence :

1. Calculer la DFT des indices pairs de $f$.
2. Calculer la DFT des indices impairs de $f$.
3. Combiner en $O(N)$ les deux suivant la formule ci-dessus.

# Implémentation

$$\sqrt{N} DFT(f)[k] = \sqrt{\frac{N}{2}} \left( DFT(f_{0:2:N-2})[k] + e^{-\frac{2i\pi}{N}k} DFT(f_{1:2:N-1})[k] \right)$$

Les sous DFT sont de longueur $N/2$, donc définies pour $0 \leq k \leq N/2$. On utilise :

$$\exp(-\frac{2i\pi}{N/2} j(k+N/2)) = \exp(-\frac{2i\pi}{N/2} jk)$$

$$\exp(-\frac{2i\pi}{N}(k+N/2)) = \exp(-\frac{2i\pi}{N}k)$$

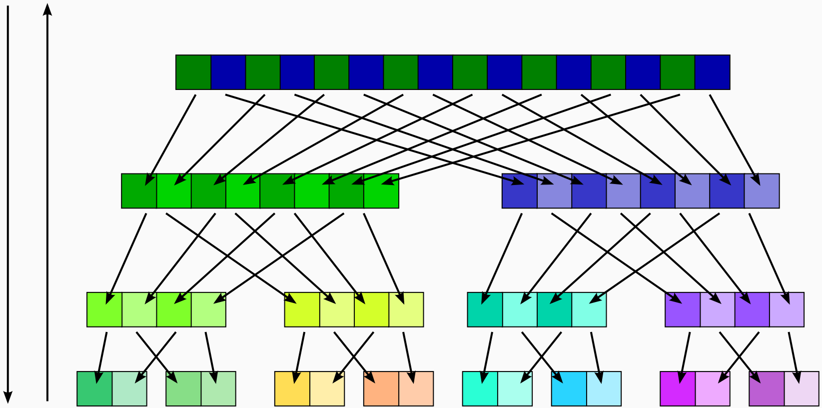
# Implémentation

$$\sqrt{N} DFT(f)[k] = \sqrt{\frac{N}{2}} \left( DFT(f_{0:2:N-2})[k] + e^{-\frac{2i\pi}{N}k} DFT(f_{1:2:N-1})[k] \right)$$

1. Calculer la DFT des indices pairs de $f$.
2. Calculer la DFT des indices impairs de $f$.
3. Combiner en $O(N)$ les deux suivant la formule ci-dessus.
   - Copier $f$ dans un tableau temporaire `buffer`. On a dans les indices pairs et impairs les résultats des sous-DFT (non normalisées).
   - Boucle de $k = 0$ à $N/2 - 1$ :

$$\begin{aligned} f[k] &\leftarrow \texttt{buffer}[2*k] + e^{-\frac{2i\pi}{N}k} \texttt{buffer}[2*k+1] \\ f[k+N/2] &\leftarrow \texttt{buffer}[2*k] - e^{-\frac{2i\pi}{N}k} \texttt{buffer}[2*k+1]. \end{aligned}$$

# Implementation



Calcul de la DFT au niveau le plus bas

# En pratique

- Le facteur $t_k = e^{-\frac{2i\pi}{N}k}$ est appelé *twiddle*. Calcul rapide par la relation de récurrence de suite géométrique $t_{k+1} = r\, t_k$, de raison $r = e^{-\frac{2i\pi}{N}}$ pré-calculée.
- On alloue une seule fois `buffer` et on le passe dans les arguments de la fonction récursive.

# Dérivation

Il est possible de définir un équivalent de dérivation pour la DFT.

Soit $e_j(x) = \frac{1}{\sqrt{N}} \exp(\frac{2i\pi}{N} jx)$. Pour $k \in \mathbb{N}$ : $e_j(k) = e_j[k]$

Soit $f$ la fonction périodique :

$$f(x) = \sum_j DFT(f)[j] e_j(x),$$

et $f(k) = f[k]$ :

$$f'(x) = \sum_j DFT(f)[j] e_j'(x) = \sum_j \frac{2i\pi j}{N} DFT(f)[j] e_j(x).$$

Puis :

$$DFT(f')[j] = \frac{2i\pi j}{N} DFT(f)[j].$$

Utile pour résoudre certaines EDP.

# Dérivation

Pour les fonctions réelles : il faut que $f'$ soit réelle.

$$DFT(f')[j] = \begin{cases} \frac{2i\pi}{N} j \, DFT(f)[j] & \text{pour } 0 \leq j < N/2 \\ 0 & \text{pour } j = N/2 \\ \frac{2i\pi}{N}(j - N) \, DFT(f)[j] & \text{pour } N/2 < j < N \end{cases}$$

# Plan de la séance

# TP

TP long sur deux séances.