

cours_4

March 10, 2021

Cours 4

Le cours du jour traite des points suivants:

- Efficience d'un algorithme
- Notation de Landau
- Interprétation graphique
- Tri par fusion
- Bonus: module *timeit*

1 Efficience d'un algorithme

1.1 Taille d'un problème

L'efficacité d'un algorithme dépend généralement la taille des données en entrée:

- Taille d'un tableau, d'une liste, d'un fichier
- Longueur d'une chaîne de caractères
- Quantité d'informations (nombres de bits) pour coder une information

Il est donc pertinent d'étudier l'efficience d'un algorithme par rapport à la taille des données en entrée.

1.1.1 Cas où n est "petit"

Pour des problèmes de taille petite:

- L'architecture de la machine est prépondérante
- Les "caches" (pré-calculs) sont utilisés implicitement pour accélérer les calculs.
- La mesure du temps par *time* est relativement précise.

Il est difficile de comparer deux algorithmes dans ce cas là.

=> Mesure fine du temps d'exécution (ex.: module *timeit*).

1.1.2 Cas où n est grand

Pour des problèmes de grande taille:

- Les "caches" sont insuffisants pour accélérer les calculs.
- On doit l'évolution du nombre d'opérations élémentaires (Ops) en fonction de n :
 - Nombre de cases parcourues
 - Nombre d'échanges pour un tri en place

– Nombre de comparaisons

Comme nos algorithmes ont:

- des branchements conditionnels Si $\langle \text{condition} \rangle$ alors $\langle \text{instructions alors} \rangle$
Sinon $\langle \text{instructions sinon} \rangle$
- des boucles “Tant que” Tant que $\langle \text{condition} \rangle$ faire $\langle \text{instructions} \rangle$

le nombre d’opérations élémentaires $\text{Ops}(n)$ varie même pour n fixé:

- Suivant si l’on effectue $\langle \text{condition} \rangle + \langle \text{instructions alors} \rangle$ ou $\langle \text{condition} \rangle + \langle \text{instructions sinon} \rangle$
- Suivant le nombre d’itérations de $\langle \text{condition} \rangle + \langle \text{instructions} \rangle$

On parlera donc de:

- Borne sup : “pire des cas”, ie trouver une configuration de la donnée qui maximise $\text{Ops}(n)$.
- Borne inf : “meilleur des cas”, ie trouver une configuration de la donnée qui minimise $\text{Ops}(n)$.

Pour n fixé:

$$\text{Meilleur} \leq \text{Raliste} \leq \text{Pire}$$

On regarde ainsi l’évolution de ses bornes (cet encadrement) en fonction de n .

=> Les notations de *Landau*.

Elles vont nous permettront de comparer les algorithmes.

1.2 Notations de Landau

1.2.1 Notation ‘grand o’ pour la borne sup (Pire des cas)

$$\mathcal{O}(f(n)) := \{g : \mathbb{N} \rightarrow \mathbb{R}^* \mid \exists c \in \mathbb{R}^*, n_0 \in \mathbb{N} \setminus \forall n \geq n_0, g(n) \leq c f(n)\}$$

$\text{Ops}(n) \in \mathcal{O}(n)$ dire “Ops(n) est en grand o de n”

=> Borne supérieure asymptotique sur nombre d’opérations (au coef. mult. c près)

$$\begin{array}{ll} \text{Ex : } \text{Ops}(n) = 10n^2 - 5n + 2 & \begin{array}{l} \text{— } \text{Ops}(n) \in \mathcal{O}(n^2) \\ \text{— } \text{Ops}(n) \notin \mathcal{O}(n) \\ \text{— } \text{Ops}(n) \in \mathcal{O}(n^3) \end{array} \end{array}$$

1.2.2 Notation ‘grand omega’ pour la borne inf (Meilleur des cas)

$$\Omega(f(n)) := \{g : \mathbb{N} \rightarrow \mathbb{R}^* \mid \exists c \in \mathbb{R}^*, n_0 \in \mathbb{N} \setminus \forall n \geq n_0, g(n) \geq c f(n)\}$$

$Ops(n) \in \Omega(n)$ dire "Ops(n) est en grand omega de n"

\Rightarrow Borne inférieure asymptotique sur nombre d'opérations (au coef. mult. c près)

$$\begin{array}{ll} \text{Ex : } Ops(n) = 10n^2 - 5n + 2 & \begin{array}{l} \text{--- } Ops(n) \in \Omega(n^2) \\ \text{--- } Ops(n) \in \Omega(n) \\ \text{--- } Ops(n) \notin \Omega(n^3) \end{array} \end{array}$$

1.2.3 Notation ‘grand theta’

$$\Theta(f(n)) \sim \Omega(f(n)) \bigcap \mathcal{O}(f(n))$$

$$\begin{aligned} \Theta(f(n)) := & \{g : \mathbb{N} \rightarrow \mathbb{R}^* \mid \exists (c_1, c_2) \in \mathbb{R}^* \times \mathbb{R}^*, n_0 \in \mathbb{N} \\ & \text{tel que } \forall n \geq n_0, \quad c_1 f(n) \leq g(n) \leq c_2 f(n)\} \end{aligned}$$

$Ops(n) \in \Theta(n)$ dire "Ops(n) est en grand theta de n"

$\Rightarrow Ops(n)$ est encadrée par $f(n)$ "de l'ordre exact de"

$$\begin{array}{ll} \text{Ex : } Ops(n) = 10n^2 - 5n + 2 & \begin{array}{l} \text{--- } Ops(n) \in \Theta(n^2) \\ \text{--- } Ops(n) \notin \Theta(n) \\ \text{--- } Ops(n) \notin \Theta(n^3) \end{array} \end{array}$$

1.2.4 Echelle de comparaison (voir code)

notation	grandeur au plus
$O(1)$	module majoré par une constante
$O(\log(n))$	logarithmique
$O(n)$	linéaire
$O(n \log(n))$	quasi-linéaire
$O(n^2)$	quadratique
$O(n^c)$	polynomial d'ordre c
$O(c^n)$	exponentiel

1.2.5 Applications aux algorithmes déjà rencontrés

- Calcul du min/max: $\Theta(n)$
- Recherche d'un élément: $\Omega(1)$ et $O(n)$, en moyenne : $n/2$
- Tri par sélection: $\Theta(n^2)$
- Tri par insertion: $\Omega(n)$ et $O(n^2)$, en moyenne : $n^2/4$

2 Stratégie Diviser pour Régner

- Simplifier un problème en tâches plus simples
- Fusionner les résultats

2.1 Tri par fusion (merge sort)

Principe:

- T est un tableau de taille n
- Partitionner T en deux parties égales (taille $n/2$) -> Gauche, Droite
- Trier récursivement **Gauche** et **Droite** (parallèle possible)
- Fusionner **Gauche** et **Droite** en T

Voir Dessin.

2.1.1 Tri Fusion: partitionnement

```
[4]: def partitionnement (T):  
      mil = len(T) // 2
```

```
[2]: # remarque  
T = [1, 2, 3, 4]  
G, D = T[:2], T[2:]  
D[0] = 1000
```

```
[2]: [1, 2, 3, 4]
```

2.1.2 Tri Fusion: Fusion

Principe:

- Parcourir simultanément **Gauche** et **Droite**
- Recopier l'élément minimal entre les deux valeurs courantes
- Augmenter l'indice sur le tableau qui contenait ce minimum.
- Continuer jusqu'à avoir recopier **Gauche** et **Droite**

Algorithme 5: Fusion

Données : $G[1..g]$ et $D[1..d]$ deux tableaux *triés*, $T[1..g+d]$

Résultat : $T[1..g+d]$ est trié et contient les éléments de G et D

```
1  $i_t \leftarrow 1; i_g \leftarrow 1; i_d \leftarrow 1;$ 
2 tant que  $i_g \leq g$  et  $i_d \leq d$  faire
3   | si  $G[i_g] \leq D[i_d]$  alors
4   |   |  $T[i_t] \leftarrow G[i_g]; i_g \leftarrow i_g + 1$ 
5   | sinon
6   |   |  $T[i_t] \leftarrow D[i_d]; i_d \leftarrow i_d + 1$ 
7   | fin
8   |  $i_t \leftarrow i_t + 1;$ 
9 fin
10 tant que  $i_g \leq g$  faire  $T[i_t] \leftarrow G[i_g]; i_t \leftarrow i_t + 1; i_g \leftarrow i_g + 1;$ 
11 tant que  $i_d \leq d$  faire  $T[i_t] \leftarrow D[i_d]; i_t \leftarrow i_t + 1; i_d \leftarrow i_d + 1;$ 
```

Voir codes:

- Algorithme
- Vérification

2.1.3 Tri Fusion: Preuves

- Terminaison: Fin de la récursivité
- Correction de l'algorithme = Correction de la fusion
- Efficience: $\text{Ops}(n) = \text{NbComp}(n) \in \Theta(n \log n)$

2.1.4 Tri Fusion: Caractéristiques

- Tri par comparaison
- Tri stable
- Tri qui n'est pas en place ... > Utilisation de la mémoire: $\Theta(n)$
- Donc un tri extrêmement efficace

[]: