

Big Data

TD 4 : Complexité algorithmique BUT 3

Guillaume Metzler et Antoine Rolland

Institut de Communication (ICOM)

Université de Lyon, Université Lumière Lyon 2

Laboratoire ERIC UR 3083, Lyon, France

guillaume.metzler@univ-lyon2.fr; antoine.rolland@univ-lyon2.fr

Les exercices de cette fiche permettent de mettre en avant l'importance de la façon dont les traitements statistiques doivent être effectués. On aborde la notion de complexité des algorithmes développés. Cette notion est primordiale en milieu industrielle, notamment quand il s'agit de développer des solutions qui *passent à l'échelle*.

Exercice 1 : Un peu de complexité

1. Calculez le nombre minimal d'opérations élémentaires pour calculer la moyenne d'un n échantillon de variables quantitatives continues.

La formule de la moyenne est :

$$\frac{1}{n} \sum_{i=1}^n x_i.$$

Donc, pour la somme, il faut au moins $n - 1$ opérations (en stockant par exemple dans la première valeur) mais plus souvent n opérations (en créant une autre valeur) et on finit par une division soit $n + 1$ opérations en tout.

2. Calculez le nombre minimal d'opérations élémentaires pour calculer la moyenne d'un n échantillon de variables quantitatives discrètes avec K modalités effectives si nous connaissons le tri à plat.

Dans le cas d'un tri à plat avec K modalités effectives, nous devons utiliser la formule :


$$\frac{1}{n} \sum_{k=1}^K a_k \times n_k.$$

Nous commençons par faire toutes les multiplications soient K opérations puis la somme des valeurs $K - 1$ si on stocke la première valeur. Enfin, il faut diviser par n , cela fait une

opération si on connaît la valeur mais K opérations si nous devons le calculer. Dans le pire des cas, il faut donc $3K - 1$ opérations.

3. Comment appelle-t-on la complexité du calcul d'une moyenne ?

Pour le cas continu, nous avons une complexité linéaire en n et dans le cas discret, nous avons une complexité linéaire en K .

4. Donnez la complexité de chacun des algorithmes suivants codés en .

```
## Algorithme 1 ##  
  
x <- rnorm(n)  
xbar<-mean(n)  
s<-0  
for (i in 1:n){  
  s<-s+(x[i]-xbar)^2  
}  
s<-sqrt(s/n)  
  
## Algorithme 2 ##  
x <- rnorm(n)  
sd(x)*(n-1)/n  
  
## Algorithme 3 ##  
x <- rnorm(n)  
s<-0  
for (i in 1:n){  
  xbar<-mean(n)  
  s<-s+(x[i]-xbar)^2  
}  
s<-sqrt(s/n)
```

La complexité pour simuler un vecteur gaussien est linéaire en n , de même pour la moyenne. L'affectation est constante et au sein de chaque étape de la boucle. Du coup, la boucle d'une complexité constante est linéaire. Enfin, la division et la racine carré est en complexité constante. Au final, l'algorithme 1 a une complexité linéaire en n .

Pour l'algorithme 2, la fonction `sd` fait la même chose que l'algorithme 1 donc c'est la même complexité.

Pour l'algorithme 3, la différence avec l'algorithme 1 est que le calcul de la moyenne est refait à chaque itération de la boucle. Du coup, la complexité est linéaire en n au sein de chaque itération donc, au final, elle est quadratique en n .

5. Étant donné un n -échantillon, quelle est la complexité pour extraire toutes les parties de cet échantillon (c'est-à-dire, l'ensemble vide, tous les individus, tous les couples, tous les triplets et ainsi de suite jusqu'à l'échantillon complet) ?

Pour calculer l'ensemble des parties, nous avons le choix de prendre ou ne pas prendre chaque élément. Donc, à chaque fois, nous avons 2 possibilités qui se multiplient entre elles soit 2^n ensembles. La complexité est donc exponentielle.

On pourrait raisonner à l'aide des coefficients binomiaux pour le choix du nombre d'éléments parmi un ensemble de n éléments et utiliser la formule du binôme de Newton.

6. On revient maintenant sur l'algorithme de l'Analyse en Composantes Principales appliquée à une matrice de données $mX \in \mathcal{M}_{n,p}(\mathbb{R})$, dont les différentes étapes sont données ci-dessous :

- (a) Centrage et réduction des variables de la matrice \mathbf{X} .

$$\mathcal{O}(np).$$

- (b) Calcul du produit $\mathbf{X}^\top \mathbf{X}$.

$$\mathcal{O}(np^2).$$

- (c) Recherche des valeurs propres de la matrice $\mathbf{X}^\top \mathbf{X}$ ainsi que les vecteurs propres associés.

$$\mathcal{O}(p^3).$$

C'est en fait la complexité du pivot de gauss pour se ramener à une matrice triangulaire.

- (d) Projection sur les sous-espaces propres des variables en calculant les $\mathbf{X}u_k$, $k = 1, \dots, s$.

$$\mathcal{O}(snp).$$

- (e) Projection des individus sur les sous-espaces en calculant les $\frac{1}{\lambda_s} \mathbf{X}^\top \mathbf{X}u_k$, $k = 1, \dots, s$.

$$\mathcal{O}(snp^2).$$

7. Déterminer la complexité de chaque étape ainsi que la complexité globale de la procédure. Que se passe-t-il si p est plus petit que n ?

La complexité est donc de $\mathcal{O}(np^2 + p^3)$. Si $p \leq n$, la complexité est réduite à $\mathcal{O}np^2$.

Exercice 2 : Suite de Fibonacci

L'histoire des Mathématiques est parfois surprenante, et décidément toujours inattendue. Le vieux nombre d'or (qui apparaît dans la suite), à l'origine géométrique, s'apparenta des siècles plus tard avec des fractions issues d'une suite purement arithmétique. L'artisan de cette union fut le plus remarquable mathématicien du Moyen Âge, Leonardo Pisano, plus connu sur le nom de Fibonacci¹. Le plus célèbre de tout les problèmes qui fait apparaître ce nombre d'or se trouve certainement dans le **Livre de l'abaque**. Il s'agit du fameux problème des lapins, dont la solution est la suite aujourd'hui

1. **Leonardo Pisano, Fibonacci (1170 - 1250)**. Il naquit à Pise en 1170. Son surnom renseigne sur son origine familiale : Fibonacci signifie tout simplement «fils de Bonacci» (figlio di Bonacci). Cependant, ce nom est d'origine moderne ; on ne dispose d'aucune preuve permettant d'affirmer qu'il était connu sous le patronyme de Fibonacci. Il s'initia aux Mathématiques à partir de la comptabilité, car son père était un marchand italien qui avait des activités commerciales internationales. Rapidement, Leonardo montra un vif intérêt pour les mathématiques qui allait bien au-delà de leurs applications mercantiles. Ses voyages marchands en Afrique du Nord lui offrirent l'opportunité de s'initier aux mathématiques arabes aux côtés de maîtres musulmans. Il connut ainsi le système de numérotation arabo-hindou et en comprit immédiatement les énormes avantages. En Europe, il en devint le défenseur le plus zélé et tenta de le vulgariser. C'est à lui que nous devons apparition dans notre culture.

connue sous le nom de Fibonacci.

Le problème est posé de la façon suivante : combien de couples de lapins aurons-nous à la fin de l'année si nous commençons avec un couple qui engendre chaque mois un autre couple qui procrée à son tour au bout de deux mois de vie ?

L'objectif de cet exercice est alors d'étudier les solutions de ce problème et plus largement la suite de Fibonacci $(F_n)_{n \in \mathbb{N}}$ définie par $F_0 = 1$ et $F_1 = 1$ et pour tout $n \geq 1$.

$$F_{n+2} = F_{n+1} + F_n.$$

Plus précisément, on s'intéressera à des algorithmes permettant de déterminer les différents termes de cette suite, avec des complexités différentes.

1. Une version récursive de l'algorithme est donnée par la procédure suivante

Algorithm 1: Une approche récursive

Paramètre : Un entier $n \geq 0$

Sortie : Valeur de F_n

Fibo(n)

1: **if** $n \leq 1$ **then**

2: retourner n

3: **else**

4: retourner Fibo($n - 1$) + Fibo($n - 2$)

5: **end if**

Expliquer le fonctionnement de l'algorithme et évaluer sa complexité.

Évaluons le nombre d'appel de Fibo(n) nécessaire au calcul du n -ème terme de la suite de Fibonacci, notons A_n cette valeur.

Nous avons alors :

$$A_0 = 1,$$

$$A_1 = 1.$$

En effet, dans les deux premiers cas, on appelle une seule fois la fonction Fibo. De même, nous avons

$$A_2 = 3 = A_1 + A_0 + 1,$$

$$A_3 = 4 = A_2 + A_1 + 1,$$

où le terme $+1$ correspond à l'appel de la fonction Fibo pour initier le calcul et qui nécessite de alors autant d'opérations que pour le calcul des valeurs de la suite pour les deux termes précédents. Plus généralement, on a donc

$$A_{n+1} = A_n + A_{n-1} + 1.$$

On pourra également noter que le nombre d'appels à la fonction correspond au nombre de sommes effectuées pour le calcul de la $n + 1$ -ème valeur de la suite. A l'aide de notre relation de récurrence, nous avons

$$\begin{aligned}
 A_n &= A_{n-1} + A_{n-2} + 1, \\
 &\downarrow (A_n)_{n \in \mathbb{N}} \text{ est croissante} \\
 &\geq 2A_{n-2} + 1 \\
 &= 2(A_{n-3} + A_{n-4} + 1) + 1, \\
 &\downarrow (A_n)_{n \in \mathbb{N}} \text{ est croissante} \\
 &\geq 4A_{n-4} + 3, \\
 &= 4(A_{n-5} + A_{n-6} + 1) + 3, \\
 &\downarrow (F_n)_{n \in \mathbb{N}} \text{ est croissante} \\
 &\geq 8A_{n-6} + 7, \\
 &\geq \dots, \\
 &\geq 2^k A_{n-2k} + 2^k - 1.
 \end{aligned}$$

On peut continuer le processus jusqu'à ce que k atteigne la valeur $n/2$, auquel cas $n - 2k = 0$ et on retombe sur le premier terme de la suite de Fibonacci. Ainsi, pour cette valeur, nous avons

$$\begin{aligned}
 A_n &\geq 2^{n/2} A_0 + 2^{n/2} - 1, \\
 &\downarrow \text{ or } A_0 = 1 \\
 A_n &= 2^{n/2} + 2^{n/2} - 1, \\
 &\downarrow \text{ on s'affranchit du dernier terme} \\
 &\geq \sqrt{2}^n.
 \end{aligned}$$

Ce qui permet d'en déduire que le processus récursif de calcul des termes de la suite de Fibonacci est au moins exponentiel, *i.e.* $\mathcal{O}(\sqrt{2}^n)$.

2. Une autre version, dite par recensement, peut également être employée pour évaluer les termes de cette suite. La procédure est décrite ci-après :

Algorithm 2: Une approche par recensement

Paramètre: Un entier $n \geq 0$

Sortie : Valeur de F_n

Table de Stockage : T de taille n Fibo(n)

```

1: if  $n \leq 1$  then
2:   retourner  $n$ 
3: else
4:   retourner Fibo( $n - 1$ ) + Fibo( $n - 2$ )
5:    $T[n - 1] = \text{Fibo}(n - 1)$ 
6: end if

```

Expliquer le fonctionnement de l'algorithme et évaluer sa complexité.

L'approche par recensement est similaire à l'approche récursive précédemment étudiée. On va cependant tirer profit des calculs qui sont déjà effectués pour éviter de répéter des

calculs qui sont déjà effectués. Cela va nécessiter de stocker des résultats intermédiaires dans une table de taille n . Cette nouvelle procédure aura alors une complexité en mémoire en $\mathcal{O}(n)$.

Sa complexité sera également linéaire en ce qui concerne le nombre d'opérations effectuées car nous pourrions faire appel aux valeurs stockées dans une table pour éviter de calculer les différents termes. Il faudra simplement remplir un tableau avec un décalage à chaque fois et ensuite parcourir les éléments dans un tableau mais dans un ordre donné à chaque itération (coût global de n).

3. Proposer une version du calcul des termes de la suite $(F_n)_{n \in \mathbb{N}}$ par une approche itérative, de complexité linéaire en temps et constante en mémoire.

On peut aisément créer une version linéaire de cet algorithme en gardant en mémoire les valeurs précédentes. Ainsi, le nombre d'affectations évoluera linéairement au cours du temps (même si plusieurs affectations seront à effectuer au cours du temps). Le processus est décrit en Algorithme 3.

Algorithme 3: Une approche itérative

Paramètre : Un entier $n \geq 0$

Sortie : Valeur de F_n

Fibo(n)

```
1: if  $n \leq 1$  then
2:   retourner  $n$ 
3: else
4:    $u = 1$ 
5:    $v = 1$ 
6:   for  $k = 2$  à  $n$  do
7:      $T$  prend la valeur  $u + v$ 
8:      $v$  prend la valeur  $T$ 
9:      $u$  prend la valeur  $v$ 
10:  end for
11:  retourner la valeur de  $T$ 
12: end if
```

Cet algorithme effectue une addition et 3 affectations par passage dans la boucle (complexité linéaire).

Remarque Il est possible de calculer les différents termes de cette suite à l'aide d'une procédure de complexité logarithmique ($\log_2(n)$) en considérant la matrice

$$\mathbf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

On pourrait même trouver une procédure de complexité constante ! Mais la solution nécessite de résoudre une petite équation.