DUT 2 Informatique – S3
M3103 – Algorithmique avancée

CM 01 – Introduction à la complexité algorithmique

Équipe pédagogique : Camille Kurtz, Jacques Alès-Bianchetti, Simon Fuhlhaber

`prénom.nom@parisdescartes.fr`

17 août 2016

USPC
Université Sorbonne
Paris Cité

UNIVERSITÉ
PARIS DESCARTES
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

Plan

Plan

Introduction

Définition

Qu'est ce que la théorie de la complexité ?

- C'est l'étude de la quantité de ressources (**en temps et espace**) nécessaire pour la résolution de problèmes au moyen de l'exécution d'un algorithme
- Il s'agit d'étudier la difficulté intrinsèque de problèmes posés mathématiquement

A quoi cela peut servir ?

1. Comparer les performances de 2 algos traitant le même problème
2. Savoir si le code passe à l'é&chelle des masses de données (big data)

L'analyse d'un algorithme peut s'avérer difficile. Il est donc nécessaire de se donner des outils mathématiques pour parvenir à nos fins ☺

Comment mesurer la « complexité » ?

Le coût d'exécution d'un algorithme dépend

- de la machine sur laquelle s'exécute l'algorithme
- de la traduction de l'algorithme en langage exécutable par la machine

Nous ferons ici abstraction de ces deux facteurs, pour nous concentrer sur :

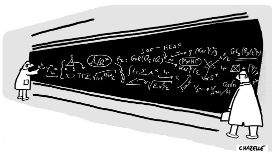
⇒ **le coût des actions résultant de l'exécution de l'algorithme, en fonction de la taille *n* des données traitées**

Complexité temporelle

Temps CPU/GPU consommé par l'algorithme

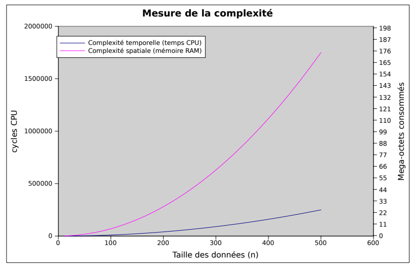
Complexité spatiale

Ressources mémoire utilisées par l'algorithme



L'allocation de cases mémoire prend du temps

Comment mesurer la « complexité » ?



Remarques

- On s'intéressera en général à la **complexité temporelle**, mais les mêmes notions permettent de traiter de la complexité spatiale
- Pour mesurer le temps d'exécution en fonction de la taille *n* des données, **il faut se donner une unité de mesure**. On va considérer que les opérations élémentaires exécutées par une machine (opérations arithmétiques, logiques, affectations) s'effectuent en **temps constant**
- L'analyse consiste alors à exprimer le nombre d'opérations effectuées comme une fonction de la taille n des données

Exemple concret

Évaluation du nombre d'opérations d'un algorithme simple

- **On considère que l'exécution de la ligne i prendra donc un temps c_i**

Algorithm 1 Calcul de la somme $\sum_{i=1}^{n} \sum_{j=1}^{i} (i+j)$

```
procedure CALCUL_SOMME(n)                 ▷ coût   fois
    somme ← 0 . . .                          c2      1
    for i ← 1, n do . . .                    c3      n
        for j ← 1, i do . . .                c4      SUM(1:n)
            somme ← somme + i + j . . .      c5      SUM(1:n)
        end for
    end for
    return somme . . .                       c8      1
end procedure
```

Le coût de l'exécution de l'algorithme dépend donc de n. On a donc : Un algorithme quadratique car l'ordre de frandeur de la complexité est n^2

Plan

Complexité asymptotique

Petit problème de complexité

Soit 2 algorithmes A et B traitant le même problème. La complexité de A est $100n$ tandis que celle de B est n^2. Quel est l'algorithme le plus efficace ?

Solution : Le rapport des complexités de B à A est égal à $n/100$. Donc :

- Pour n < 100 B est plus efficace
- Pour n = 100, A et B ont la même efficacité
- Pour n > 100 A est plus efficace

Notons que plus $n \nearrow$, plus A est efficace devant B.

Remarques

- Si les tailles des données sont « petites », la plupart des algorithmes résolvant le même problème se valent
- C'est le comportement de la complexité d'un algorithme **quand la taille des données devient grande** qui est important :

  Que se passe-t-il quand n tend vers l'infini ?

- On appelle ce comportement la complexité asymptotique

Comparaison asymptotique

Définition

La **comparaison asymptotique** est une méthode consistant à étudier le comportement d'une fonction au voisinage d'un point (ou en l'infini), en regard du comportement d'une autre fonction réputée « connue », souvent choisie sur une échelle de référence

Comment faire ?

Soit f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{N}$.

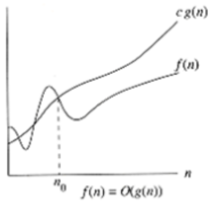
- **Comment montrer que f est négligeable devant g, ou que g est prépondérante devant f, au voisinage de l'infini ?**
- Pour décrire formellement cette « distance » entre deux fonctions, . . . on va regarder le comportement du quotient f / g
- On utilise pour cela les **notations de Landau**

Edmund Landau (1877 – 1938)

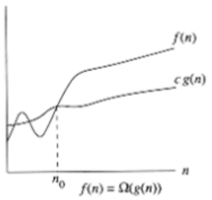
Notations de Landau

Définitions

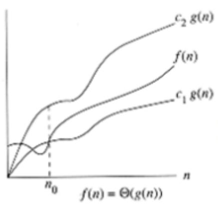
- **On dira que $f = \mathcal{O}(g)$** (prononcé f est en grand o de g) ssi il existe $c > 0$ et $n_0 \in \mathbb{N}$ tel que $\forall n \geqslant n_0 : f(n) \leqslant c \cdot g(n)$
- **On dira que $f = \Omega(g)$** (prononcé f est en oméga de g) ssi il existe $c > 0$ et $n_0 \in \mathbb{N}$ tel que $\forall n \geqslant n_0 : f(n) \geqslant c \cdot g(n)$
- **On dira que $f = o(g)$** (prononcé f est en o de g) ssi $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$
- **On dira que $f = \Theta(g)$** (prononcé f est en thêta de g) ssi $f = \mathcal{O}(g)$ et $g = \mathcal{O}(f)$. On remarquera que $f = \Theta(g)$ ssi $g = \Theta(f)$



$f = \mathcal{O}(g)$



$f = \Omega(g)$



$f = \Theta(g)$

Notations de Landau

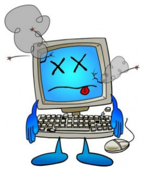
Pour faire plus simple . . .

1. La notation $\mathcal{O}$ permet majorer le comportement d'une fonction
2. La notation Θ permet d'éviter d'avoir une majoration trop grossière
3. La notation Ω signifie .qu'une fonction croit au moins aussi vite qu'une autre

Voici quelques exemples de comparaisons de fonctions :

1. $n^2 + 3n + 1 = \Theta(n^2) = \Theta(50n^2)$
2. $n/\log(n) = \mathcal{O}(n)$
3. $50n^{10} = \mathcal{O}(n^{10,00001})$
4. $2^n = \mathcal{O}(exp(n))$
5. $n/\log(n) = \Omega(\sqrt{n})$
6. $log_2(n) = \Theta(log(n)) = \Theta(ln(n))$

Notations de Landau



Remarques

- Comme on ne manipule que des fonctions de $\mathbb{N}$ dans $\mathbb{N}$ et que l'on s'intéresse uniquement à leur comportement à l'infini, **on peut sommer les $\mathcal{O}$ et Θ sans risque**
- Intuitivement, la notation Θ revient à oublier le coeff multiplicatif constant de g
- On préférera utiliser en général la notation $\Theta(f(n))$ qui indique que la fonction de coût réelle est **du même ordre de grandeur** que la fonction f

Plan

Classement des familles

- **Grandes familles de complexité** et ordres de grandeur du temps nécessaire à l'exécution des algorithmes de ces familles.

			Taille des données		
Famille	**Notation**	**Exemples**	$n = 10^2$	$n = 10^3$	$n = 10^6$
Algo. constants	$\Theta(1)$	Échange 2 valeurs	10 ns	10 ns	10 ns
Algo. logarithmiques	$\Theta(\log n)$	Rech. dichotomique	25 ns	30 ns	60 ns
Algo. linéaires	$\Theta(n)$	Rech. séquentielle	1 μs	10 μs	10 ms
Algo. quasi-linéaires	$\Theta(n \log n)$	Tri par fusion	1 μs	10 μs	10^4 μs
Algo. quadratiques	$\Theta(n^2)$	Tri par sélection	100 μs	10 ms	2,8 h
Algo. cubiques	$\Theta(n^3)$	Produit 2 matrices	75 ms	10 s	16 ans
Algo. polynomiaux	$\Theta(n^p)$				
Algo. exponentiels	$\Theta(a^n)$	Suite Fibonacci			
Algo. factoriels	$\Theta(n!)$	Voyageur commerce	10^{80}	...	...
			Temps d'exécution		

Petits problèmes

Quel est le temps CPU nécessaire pour ...

1. calculer le produit de 2 matrices quand $n = 10^6 = 1000000$?
2. calculer le voyageur de commerces quand $n = 10^2 = 100$?

...

Ordres de grandeurs de complexités

Quelques chiffres ...

La complexité asymptotique c'est bien mais pour une complexité donnée, il est important d'avoir une idée **des ordres de grandeurs** des données que l'algorithme pourra traiter.

Voici quelques chiffres pour se faire une idée :

- effectuer 2^{30} (environ un milliard) opérations binaires sur un PC standard prend ...
- le **record actuel de puissance de calcul** fourni pour résoudre un problème donné est de l'ordre de 2^{30} opérations bianires
- aujourd'hui en cryptographie, on considère qu'un problème dont la résolution nécessite 2^{80} opérations binaires est impossible à résoudre
- une complexité de 2^{128} opérations binaires sera a priori toujours impossible à résoudre

Plan

Au meilleur du pire

Petit problème de complexité

Un algorithme de recherche d'une valeur dans un tableau T peut s'arrêter dès qu'il a trouvé une occurrence de cette valeur.
Comment évaluer sa complexité *a priori* ?

Solution : Il faut distinguer plusieurs cas de complexité :

1. **au meilleur des cas** (temps CPU faible si la valeur est au début de T)
2. **au pire des cas** (temps CPU élevé si la valeur se trouve à la fin de T)
3. **en moyenne**

Complexité variable : dépendance aux données

- Le temps d'exécution **dépend** de la nature des données .
- Nous nous intéresserons en général à **la complexité au pire des cas**, qui est une manière pessimiste d'ignorer cette dépendance

  Exemple : on évaluera le temps d'exécution en supposant qu'il faut parcourir tout le tableau pour trouver la valeur cherchée

Complexités en temps d'un algorithme

Soit D_n l'ensemble des données possibles de taille n et soit $C(d)$ le coût d'exécution de l'algorithme sur la donnée d de taille n.

Complexité au meilleur

La complexité au meilleur, ou complexité dans le meilleur cas, est **le plus petit nombre d'opérations** qu'aura à exécuter l'algorithme sur un jeu de données de taille n :

$$T_{\min}(n) = \min_{d \in D_n} C(d)$$

Avantage

C'est une borne inférieure de la complexité de l'algorithme

Notions Notations asymptotiques Familles de complexité **Différents cas** Exemple Pour aller plus loin

Complexités en temps d'un algorithme

Soit D_n l'ensemble des données possibles de taille n et soit $C(d)$ le coût d'exécution de l'algorithme sur la donnée d de taille n.

Complexité au pire

La complexité au pire, ou complexité dans le pire cas (worst-case en anglais), est **le plus grand nombre d'opérations** qu'aura à exécuter l'algorithme sur un jeu de données de taille n :

$$T_{\max}(n) = \max_{d \in D_n} C(d)$$

Avantage

Il s'agit d'un maximum, et l'algorithme finira toujours avant d'avoir effectué $T_{\max}(n)$ opérations

Inconvénient

Cette complexité peut ne pas refléter le comportement « usuel » de l'algorithme, le pire cas pouvant ne se produire que très rarement, mais il n'est pas rare que la complexité au pire des cas soit également la complexité moyenne.

Complexités en temps d'un algorithme

Soit D_n l'ensemble des données possibles de taille n et soit $C(d)$ le coût d'exécution de l'algorithme sur la donnée d de taille n.

Complexité en moyenne 1/2

La complexité en moyenne est la **moyenne des complexités** de l'algorithme sur des jeux de données de taille n :

$T_{moy}(n) =$ ...SUM(min d de Dn . C(d))

où $Pr(d)$ est la probabilité d'avoir la donnée d en Input de l'algorithme

Avantage

Elle reflète le ... comportement général de l'algorithme si les cas extrêmes sont rares ou si la complexité varie peu en fonction des données

Inconvénient

En pratique la complexité peut être nettement plus importante que la complexité en moyenne, dans ce cas **la complexité en moyenne** ne... donnera pas une bonne indication du comportement de l'algorithme

Complexités en temps d'un algorithme

Soit D_n l'ensemble des données possibles de taille n et soit $C(d)$ le coût d'exécution de l'algorithme sur la donnée d de taille n.

Complexité en moyenne 2/2

Si toutes les configurations des données de taille n fixée sont équiprobables, la complexité en moyenne s'exprime en fonction du nombre $|D_n|$ de données de taille n :

$T_{moy}(n) =$ ...1/SUM(max d de Dn . C(d))

Attention

Ce n'est pas parce qu'un algorithme est meilleur en moyenne qu'un autre en moyenne, qu'il est meilleur dans le pire des cas

En pratique ...

La complexité en moyenne est beaucoup plus difficile à déterminer que la complexité dans le pire cas parce qu'il n'est pas toujours facile de déterminer un modèle de probabilités adéquat au problème.

Complexités en temps d'un algorithme

Soit D_n l'ensemble des données possibles de taille n et soit $C(d)$ le coût d'exécution de l'algorithme sur la donnée d de taille n.

Propriété des complexités

La complexité en moyenne et les complexités extrémales vérifient la relation :

$$T_{\text{min}}(n) \leq T_{\text{moy}}(n) \leq T_{\text{max}}(n)$$

Remarques

- Si le comportement de l'algorithme ne dépend pas de la configuration des données, ces trois quantités sont confondues. Mais en général, ce n'est pas le cas et l'on ne sait pas si le coût moyen est plus proche du coût minimal ou du coût maximal (sauf si l'on sait déterminer les fréquences relatives des configurations donnant un coût minimal et celles donnant un coût maximal).
- En pratique, nous ne nous intéresserons qu'à **la complexité au pire et à la complexité en moyenne**.

Complexités en temps d'un algorithme

Soit D_n l'ensemble des données possibles de taille n et soit $C(d)$ le coût d'exécution de l'algorithme sur la donnée d de taille n.

Notion d'optimalité

Un algorithme est dit optimal si **sa complexité est la complexité minimale** parmi les algorithmes de sa classe.

Exemple :
On peut montrer que tout algorithme résolvant le problème du tri a une complexité dans le pire des cas en $\Omega(n \log n)$. Le tri par tas (heapsort) est en $\mathcal{O}(n \log n)$ dans le pire des cas : il est donc optimal

Plan

Complexité au pire, au meilleur et moyenne

	▷ coût	fois
procedure void ↩ ALGO(T[n] : tableau de 1 et de 0)		
$k \leftarrow 0$ . . .	c2	1
for $i \leftarrow 0, n-1$ **do** . . .	c3	n
if T[i]==1 **then** . . .	c4	n
$k \leftarrow 1$ . . .	c5	n
end if	c6	
end for	c7	
if k==0 **then** . . .	c8	
for $i \leftarrow 0, n-1$ **do** . . .	c9	1
for $j \leftarrow 0, n-1$ **do** . . .	c10	n
$k \leftarrow k+1$ . . .	c11	n^2
end for	c12	n^2
end for	c13	
end if	c14	
end procedure	c15	

Questions

Évaluer la complexité de cet algorithme au pire, au meilleur et en moyenne

. . .

Complexité au pire, au meilleur et moyenne

Complexité en moyenne

2^n tableaux possibles, donc 2^n configurations différentes :

- Pour $2^n - 1$ configurations (nb de tableaux contenant au moins un 1)

$$T(n) = 3n + 2$$

- Pour 1 configuration (nb de tableaux contenant que des 0),

$$T(n) = 2n^2 + 4n + 2$$

On peut ainsi évaluer la complexité moyenne comme : ...

Tmoy = 3n + 2 => Theta(n)

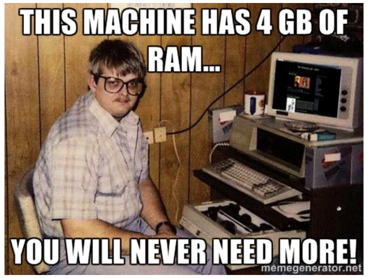
Plan

Consommation mémoire

Analyse de la complexité spatiale

- **Évaluer la quantité de mémoire nécessaire en fonction de la taille des données** $\rightarrow$ on utilise la même approche que pour évaluer la complexité temporelle de l'algorithme (en fonction de n)
- La conservation des résultats intermédiaires en mémoire permet souvent de diminuer la complexité en temps
- Mais la mémoire doit être suffisante ...

Compromis entre espace et temps



Classes de difficulté

Qu'est-ce que la classe P ?

- Un problème est dans la classe P si et seulement si c'est **un problème de décision admettant un algorithme déterministe de temps polynomial**
- Un exemple classique d'un problème de cette classe est de savoir si un graphe est connexe
- On peut aussi dire que tous les problèmes de décisions (ceux qui ont une réponse par oui ou non) qui peuvent être facilement résolus par un algorithme polynomial sont dans cette classe

Qu'est-ce que la classe NP ?

- Un problème est dans la classe NP si et seulement si c'est **un problème de décision admettant un algorithme non déterministe de temps polynomial**
- Un exemple classique d'un problème de cette classe est de savoir si un graphe est hamiltonien
- On peut aussi dire que tous les problèmes que l'on résout par un algorithme qui énumère toute les possibilités sont dans cette classe

Classes de difficulté

Qu'est-ce qu'un problème X-complet ?

Un problème est dit X-complet si et seulement s'il est :

- de classe X (P, NP,...) ;
- de type X-difficile.

Qu'est-ce qu'un problème X-difficile ?

- Un problème est dit X-difficile si et seulement **s'il est au moins aussi dur que tous les problèmes dans X, sachant que X est une classe quelconque (P, NP,..)**
- Plus concrètement on dit que l'on réduit un problème difficile en un autre plus facile de la même classe et si on arrive à résoudre le plus facile alors on peut résoudre le difficile

Notions Notations asymptotiques Familles de complexité Différents cas Exemple Pour aller plus loin

Références

Bibliographie

Des éléments de ce cours sont empruntés de [Knuth(1997), Cormen(2011), Zampieri(2013)]

T. Cormen.
Introduction à l'algorithmique.
Dunod, 2011.

D. Knuth.
The art of computer programming, vol. 1 : Fundamental algorithms.
Addison-Wesley, 1997.

K. Zampieri.
Algorithmique – complexité des algorithmes.
Unisciel algoprog, 2013.
URL http://ressources.unisciel.fr/algoprog/s34plexite/emodules/cx00macours1/co/cx00macours1.html.