Annexe: Notion de complexité

Recherche Opérationnelle et Optimisation
Master 1 I2L / ISIDIS

Sébastien Verel
verel@lisic.univ-littoral.fr
http://www-lisic.univ-littoral.fr/~verel

Université du Littoral Côte d'Opale
Laboratoire LISIC
Equipe CAMOME

Plan

1. Introduction
2. Définition de complexité
3. Calcul de complexité

Reparlons de cuisine

Problème

Convevoir un bon gâteau aux citrons

Ingrédients (pour 6 personnes) :

- 1 pâte brisée
- 150 g de sucre
- 100 g de beurre fondu
- 3 oeufs
- le jus de deux citrons

Préparation :

. Préchauffer le four à 200°C.

. Abaisser la pâte brisée.

. Battre les oeufs avec le sucre en poudre jusqu'à l'obtention d'un mélange mousseux.

. Ajouter le jus de citron.

. Ajouter le beurre fondu.

. Enfourner et laisser cuire environ 30 mn.

. La préparation doit dorer.

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?
- Est-ce que j'ai le matériel et la place pour réaliser la recette ?

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?
- Est-ce que j'ai le matériel et la place pour réaliser la recette ?

On peut même aller plus loin :

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?
- Est-ce que j'ai le matériel et la place pour réaliser la recette ?

On peut même aller plus loin :

- Est-ce la recette de marmiton est plus rapide que la recette de grand-mère ?

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?
- Est-ce que j'ai le matériel et la place pour réaliser la recette ?

On peut même aller plus loin :

- Est-ce la recette de marmiton est plus rapide que la recette de grand-mère ?
- Qu'est-ce que je peux faire comme recette avec une casserolle et une poelle ?

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?
- Est-ce que j'ai le matériel et la place pour réaliser la recette ?

On peut même aller plus loin :

- Est-ce la recette de marmiton est plus rapide que la recette de grand-mère ?
- Qu'est-ce que je peux faire comme recette avec une casserolle et une poelle ?
- Est-ce que j'ai assez de plats pour réaliser la recette pour 15 personnes ?

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?
- Est-ce que j'ai le matériel et la place pour réaliser la recette ?

On peut même aller plus loin :

- Est-ce la recette de marmiton est plus rapide que la recette de grand-mère ?
- Qu'est-ce que je peux faire comme recette avec une casserolle et une poelle ?
- Est-ce que j'ai assez de plats pour réaliser la recette pour 15 personnes ?
- Mes amis arivent dans une heure, est-ce que j'ai assez de temps pour faire 100 choux à la crème ?

Reparlons cuisine

Quand on fait modestement un peu de cuisine plusieurs questions se posent :

- Combien de temps faut-il pour préparer cette recette ?
- Est-ce que j'ai le matériel et la place pour réaliser la recette ?

On peut même aller plus loin :

- Est-ce la recette de marmiton est plus rapide que la recette de grand-mère ?
- Qu'est-ce que je peux faire comme recette avec une casserolle et une poelle ?
- Est-ce que j'ai assez de plats pour réaliser la recette pour 15 personnes ?
- Mes amis arivent dans une heure, est-ce que j'ai assez de temps pour faire 100 choux à la crème ?
- Est-ce que la recette est bonne ?...

Question cuisine

En cuisine, deux facteurs (en autres...) sont à prendre en considération :

- Le temps de conception
- L'espace nécessaire à la conception

Bien sùr, le temps et l'espace dépendent du nombre de personnes.

Regardons les problèmes en informatique où les données ont une autre saveur...

Recherche dans un tableau de nombres entiers ordonnés

Algorithme recherche1(x : entier, t : tableau d'entiers, n : entier) : booléen
début
variable i : entier
  $i \leftarrow 0$
  tant que $i < n$ et $t[i] \neq x$ **faire**
    $i \leftarrow i + 1$
  fin tant que
  retourner $i < n$
fin

Recherche dans un tableau de nombres entiers ordonnés

```
Algorithme  recherche2(x : entier, t : tableau d'entiers, a, b : entier) : booléen
variable  c : entier
début
  si a > b alors
    retourner Faux
  sinon
    c ← (a + b)/2
    si t[c] = x alors
      retourner Vrai
    sinon
      si t[c] < x alors
        retourner recherche(x, t, c+1, b)
      sinon
        retourner recherche(x, t, a, c-1)
      fin si
    fin si
  fin si
fin
```

Comparaison d'algorithmes

- Quel est l'algorithme le plus rapide ?

Comparaison d'algorithmes

- Quel est l'algorithme le plus rapide ?

Question mal posée : cela dépend du nombre de données pour chaque algorithme....

Comparaison d'algorithmes

- Quel est l'algorithme le plus rapide ?

Question mal posée : cela dépend du nombre de données pour chaque algorithme....

- Quel est l'algorithme le plus rapide pour un tableau de taille n ?

Comparaison d'algorithmes

- Quel est l'algorithme le plus rapide ?

Question mal posée : cela dépend du nombre de données pour chaque algorithme....

- Quel est l'algorithme le plus rapide pour un tableau de taille n ?

Question encore mal posée : cela dépend des machines sur lesquels chaque algorithme s'exécute....

Comparaison d'algorithmes

- Quel est l'algorithme le plus rapide ?

Question mal posée : cela dépend du nombre de données pour chaque algorithme....

- Quel est l'algorithme le plus rapide pour un tableau de taille n ?

Question encore mal posée : cela dépend des machines sur lesquels chaque algorithme s'exécute....

- Quel est l'algorithme le plus rapide s'exécutant sur une même machine de référence (?) pour un tableau de taille n ?

Comparaison d'algorithmes

- Quel est l'algorithme le plus rapide ?

Question mal posée : cela dépend du nombre de données pour chaque algorithme....

- Quel est l'algorithme le plus rapide pour un tableau de taille n ?

Question encore mal posée : cela dépend des machines sur lesquels chaque algorithme s'exécute....

- Quel est l'algorithme le plus rapide s'exécutant sur une même machine de référence (?) pour un tableau de taille n ?

Question encore mal posée : on peut imaginer des tableaux sur lesquels chaque algorithme est très rapide par rapport à l'autre.
Comparaison des temps d'exécution :

- en moyenne
- dans le pire des cas pour chaque algorithme
- dans le meilleure des cas pour chaque algorithme

Complexité spatiale et temporelle

Il existe deux mesures de l'efficacité d'un algorithme :

- Complexité spatiale : espace mémoire nécessaire pour exécuter l'algorithme
- Complexité temporelle : temps nécessaire pour exécuter l'algorithme

Ce sont des définitions informelles...

On ne s'interessera seulement qu'à la complexité temporelle, la complexité spatiale étant en général moins critique.

Pourtant, souvent expériementalement on pense à ce genre de relation :

$$C_{spatiale} C_{temporelle} = constante$$

Mesure de complexité temporelle (D. Knuth)

Pour rendre indépendant la mesure de la vitesse de la machine (en autres), il faut définir une unité de comparaison

L'idée est de définir le nombre d'opérations élémentaires en fonction de la taille des données

Objectifs :

- Estimer le cout sans exécuter l'algorithme
- comparaison d'algorithmes

Mesure de complexité temporelle (D. Knuth)

La taille des données peut être :

- la taille d'un tableau
- la grandeur des nombres
- la taille d'une liste

Les opérations élémentaires peuvent être :

- affectation
- lecture mémoire
- addition, soutraction, etc.
- test logique

Les instructions élémentaires ont souvent de coût de 1 unité.

Vitesse de processeur

source wikipedia :

Date	Nom	Transistors	Diamètre	Fréq.	Largeur
1971	4004	2300			4 bits/
1974	8080	6000	6	2MHz	8 bits/
1979	8088	29000	3	5MHz	16 bits/
1982	80286	134000	1,5	6MHz	16 bits/
1985	80386	275000	1,5	16MHz	32 bits/
1989	80486	1200000	1	25MHz	32 bits/
1993	Pentium	3100000	0,8	60MHz	32 bits/
1997	PentiumII	7500000	0,35	233MHz	32 bits/
1999	PentiumIII	9500000	0,25	450MHz	32 bits/
2000	Pentium4	42000000	0,18	1,5GHz	32 bits/
2004	Pentium4 Prescott	125000000	0,09	3,6GHz	32 bits/
2006	Core2Duo	291000000	0,065	2,4GHz	32 bits/

Vitesse de processeur

source wikipedia :

Date	Nom	largeur	MIPS
1971	4004	4 bits/ 4 bits	
1974	8080	8 bits/ 8 bits	0,64
1979	8088	16 bits/ 8 bits	0,33
1982	80286	16 bits/ 16 bits	1
1985	80386	32 bits/ 32 bits	5
1989	80486	32 bits/ 32 bits	20
1993	Pentium	32 bits/ 64 bits	100
1997	PentiumII	32 bits/ 64 bits	300
1999	PentiumIII	32 bits/ 64 bits	510
2000	Pentium4	32 bits/ 64 bits	1700
2004	Pentium4 Prescott	32 bits/ 64 bits	9000
2006	Core2Duo	32 bits/ 64 bits	22000

MIPS : Millions d'instructions par seconde

Définitions

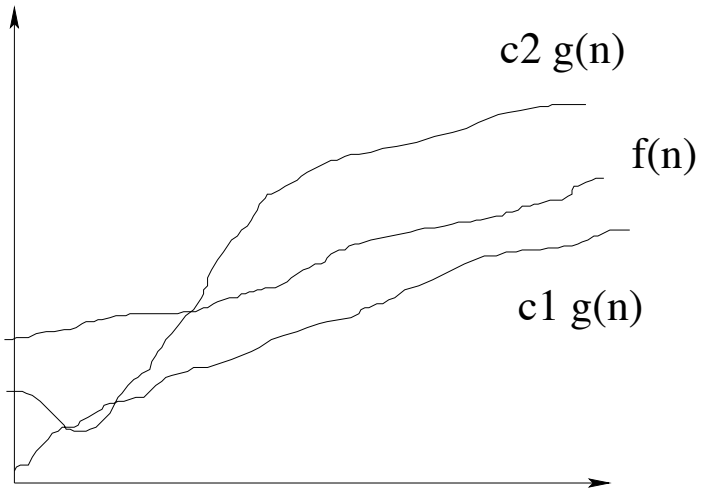
Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$.

Borne approchée asymptotique

$g(n)$ est une borne asymptotique approchée asympotiquement de $f(n)$ s'il existe deux constantes strictement positives c_1 et c_2 telles que, pour n assez grand,

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

On note alors $f(n) = \Theta(g(n))$



Définitions

Borne supérieure asymptotique

$g(n)$ est une borne supérieure approchée asympotiquement de $f(n)$ s'il existe une constante strictement positive c telles que, pour n assez grand,

$$0 \leq f(n) \leq cg(n)$$

On note alors $f(n) = \mathcal{O}(g(n))$

Définitions

Borne supérieure non asymptotiquement approchée (négligeable)

$g(n)$ est une borne supérieure non asymtotiquement approchée de $f(n)$ ou que f est négligeable devant g si

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

On note alors $f(n) = o(g(n))$

Propriétés

- si $f(n) = \Theta(g(n))$ alors $f(n) = \mathcal{O}(g(n))$
- si $f(n) = o(g(n))$ alors $f(n) = \mathcal{O}(g(n))$
- Θ et $\mathcal{O}$ sont des relations réflexives
- $\mathcal{O}$ et o sont des relations transitives
- Θ est une relation d'équivalence
- pour tout $K \in \mathbb{R}$, $\Theta(K + f(n)) = \Theta(f(n))$
- pour tout $C \in \mathbb{R}$, $\Theta(C\ f(n)) = \Theta(f(n))$
- pour tout $j < k$, $\Theta(n^k + n^j) = \Theta(n^k)$

Classes de complexité

Pour analyser la complexité, on classifie les mesures de complexité dans les catégories suivantes :

- complexité logarithmique : $\Theta(\log(n))$
  ex : recherche dichotomique dans un tableau de taille n
- complexité linéaire : $\Theta(n)$
  ex : recherche dans un tableau non ordonné de taille n
- complexité quasi-linéaire : $\Theta(n \log(n))$
  ex : tri par fusion d'un tableau de taille n
- complexité polynomiale : $\Theta(n^k)$ avec $k > 1$
  ex : multiplication de matrices
- complexité exponentielle : $\Theta(a^n)$ avec $a > 1$
  ex : tour de Hanoi

Taux de croissance

$T(n)$	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 60$
$\log(n)$	$1\mu s$	$1.3\mu s$	$1.5\mu s$	$1.6\mu s$	$1.8\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$60\mu s$
$n\log(n)$	$10\mu s$	$26\mu s$	$44\mu s$	$64\mu s$	$107\mu s$
n^2	$100\mu s$	$400\mu s$	$900\mu s$	$1.6ms$	$3.6ms$
n^3	$1ms$	$8ms$	$27ms$	$64ms$	$216ms$
2^n	$1ms$	$1s$	$18mn$	$13jour$	$366siecle$
3^n	$60ms$	$1h$	$6an$	$3900siecle$	$1.310^{13}siecle$

Eviter d'écrire des algorithmes avec des complexités exponentielles...

Effet d'une amélioration

Soit N la taille d'une instance traitable en un temps raisonnable

- Quelle taille pourra-t-on traiter lorsque les ordinateurs seront 100 et 1000 plus rapides ?

Effet d'une amélioration

Soit N la taille d'une instance traitable en un temps raisonnable

- Quelle taille pourra-t-on traiter lorsque les ordinateurs seront 100 et 1000 plus rapides ?

- Exemple 1 : $T(n) = \Theta(n^2)$
  - Aujourd'hui : $T(N) = kN^2$
  - Demain : $T^{'}(N^{'}) = \frac{T(N^{'})}{100} = k\frac{N^{'2}}{100}$ d'où $N^{'} = 10N$
- Exemple 2 : $T(n) = \Theta(2^n)$
  - Aujourd'hui : $T(N) = k2^N$
  - Demain : $T^{'}(N^{'}) = k\frac{2^{N^{'}}}{100}$ d'où $N^{'} = N + 6.67$

Effet d'une amélioration

$T(n)$	aujourd'hui	100 fois plus rapide	1000 fois plus rapide
$\log(n)$	N	N^{100}	N^{1000}
n	N	$100N$	$1000N$
n^2	N	$10N$	$32N$
n^3	N	$4.6N$	$10N$
2^n	N	$N+7$	$N+10$
3^n	N	$N+4$	$N+6$

Approximation

Pour chaque instruction de base, on peut définir un coût différent :

- 1 affectation : c_1
- 1 lecture : c_2
- 1 opération arithmétique : c_3
- 1 test : c_4

Mais on s'intéresse seulement à la classe de complexité, c'est-à-dire au coût à une constante multiplicative près.

On approxime donc le coût des instructions par un même et unique coût c unitaire.
Voire même un coût unitaire par "ligne de code"...

Conditionnel

si b **alors**
  algo de complexité C_1
sinon
  algo de complexité C_2
fin si

La complexité d'une conditionelle est :

- si b est vraie, $1 + C_1$
- si b est fausse, $1 + C_2$

Itération "pour"

pour i **de** a **à** b **faire**
  algo de complexité C_i
fin pour

La complexité d'une boucle "pour" est la somme des complexités des instructions répétées :

$$\sum_{i=a}^{b} C_i$$

Itération "pour" : cas particulier

pour i **de** 1 **à** n **faire**
  algo de complexité C
fin pour

Lorsque $\forall i \ C_i = C$ et que le nombre d'itérations est n, la complexité est de :

$$n \ C$$

Itération tant que

tant que b_i **faire**
  algo de complexité C_i
fin tant que

Soit le n le nombre d'itérations de la boucle "tant que", la complexité est alors :

$$\sum_{i=1}^{n} C_i$$

Itération tant que : remarque

tant que b_i **faire**
  algo de complexité C_i
fin tant que

Le nombre d'itérations dépend souvent des données.
On utilise alors les notions de :

- complexité dans le pire des cas : lorsque le nombre d'itération sera maximal
- complexité moyenne : nombre d'itérations moyen
- complexité dans le meilleur des cas : lorsque le nombre d'itération sera minimal

Exemple complet : recherche dans un tableau non ordonné

Algorithme recherche(x : entier, t : tableau d'entiers, n : entier) : booléen
début
variable i : entier
  $i \leftarrow 0$
  tant que $i < n$ et $t[i] \neq x$ **faire**
    $i \leftarrow i + 1$
  fin tant que
  retourner $i < n$
fin

Calcul de la complexité (temporelle)

Soit $T(n)$ la complexité pour un tableau de taille n.

Calcul de la complexité (temporelle)

Soit $T(n)$ la complexité pour un tableau de taille n.

- Une affectation au début et un test à la fin :

$$2c$$

Calcul de la complexité (temporelle)

Soit $T(n)$ la complexité pour un tableau de taille n.

- Une affectation au début et un test à la fin :

$$2c$$

- A chaque itération : 2 tests et 1 affectation

Calcul de la complexité (temporelle)

Soit $T(n)$ la complexité pour un tableau de taille n.

- Une affectation au début et un test à la fin :

$$2c$$

- A chaque itération : 2 tests et 1 affectation

$$3c$$

- Nombre d'itérations :

Calcul de la complexité (temporelle)

Soit $T(n)$ la complexité pour un tableau de taille n.

- Une affectation au début et un test à la fin :

$$2c$$

- A chaque itération : 2 tests et 1 affectation

$$3c$$

- Nombre d'itérations :
  - Au mieux : 1
  - Au pire : n
  - en moyenne : $n/2$

Calcul de la complexité (temporelle)

- Complexité au mieux : $T(n) = 5$

Calcul de la complexité (temporelle)

- Complexité au mieux : $T(n) = 5$
  $T(n) = \Theta(1)$: complexité constante
- Complexité en moyenne : $T(n) = 2 + 3\frac{n}{2}$

Calcul de la complexité (temporelle)

- Complexité au mieux : $T(n) = 5$
  $T(n) = \Theta(1)$: complexité constante
- Complexité en moyenne : $T(n) = 2 + 3\frac{n}{2}$
  $T(n) = \Theta(n)$: complexité linéaire
- Complexité au pire : $T(n) = 2 + 3n$

Calcul de la complexité (temporelle)

- Complexité au mieux : $T(n) = 5$
  $T(n) = \Theta(1)$: complexité constante
- Complexité en moyenne : $T(n) = 2 + 3\frac{n}{2}$
  $T(n) = \Theta(n)$: complexité linéaire
- Complexité au pire : $T(n) = 2 + 3n$
  $T(n) = \Theta(n)$: complexité linéaire

Complexité d'algorithmes récursifs

- Le calcul de la complexité d'un algorithme récursif conduit souvent à l'écriture d'un formule de récurrence.

Complexité d'algorithmes récursifs

- Le calcul de la complexité d'un algorithme récursif conduit souvent à l'écriture d'un formule de récurrence.
- Cette récurrence est soit une égalité soit une inégalité

Exemple du tri fusion

- Problème : Trier un tableau de données de taille n
- Méthode : "diviser pour régner"

Exemple tri par fusion

- Diviser : diviser la séquence de n éléments à trier en 2 sous-séquences de $n/2$ éléments chacune
- Régner : trier les 2 sous-séquences à l'aide du tri fusion (appel récursif)
- Combiner : fusionner les 2 sous-séquences triées pour produire la séquence complètement triée

Coût de la fusion

Algorithme fusion(t : tableau, p,q,r : entier) : rien

Entrée :

- t est supposé trié entre les indices p et q
- t est supposé trié entre les indices $q + 1$ et r

Sortie :

- t trié entre les indices p et r

La complexité de *fusionner* est en $\Theta(n)$.

Algorithme de tri par fusion

Algorithme triFusion(t : tableau, p,r : entier) : rien

Algorithme de tri par fusion

Algorithme triFusion(t : tableau, p,r : entier) : rien
début
variable q : entier
  si $p < r$ **alors**
    q $\leftarrow \frac{p+r}{2}$

Algorithme de tri par fusion

Algorithme triFusion(t : tableau, p,r : entier) : rien
début
variable q : entier
  si $p < r$ **alors**
    q $\leftarrow \frac{p+r}{2}$
    triFusion(t, p, q)
    triFusion(t, q+1, r)

Algorithme de tri par fusion

Algorithme triFusion(t : tableau, p,r : entier) : rien
début
variable q : entier
  si $p < r$ **alors**
    q $\leftarrow \frac{p+r}{2}$
    triFusion(t, p, q)
    triFusion(t, q+1, r)
    fusion(t, p, q, r)
  fin si
fin

Complexité temporelle

$T(n)$ complexité temporelle pour un tableau de taille n

- Diviser : $\Theta(1)$, calcul de q
- Régner : $2T(\frac{n}{2})$ ou plus précisement $2T(E(\frac{n}{2}))$
- Combiner : $\Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & si\ \ n = 1 \\ 2T(E(\frac{n}{2})) + \Theta(n) & si\ \ n > 1 \end{cases}$$

Calcul final

Par récurrence, on peut montrer que (voir tableau) :

$$T(n) \leq cE(\frac{n}{2})\log(E(\frac{n}{2}))$$

$$T(n) = \mathcal{O}(nlog(n))$$

Généralisation

Formule de récurrence de la forme :

$$T(n) = aT(\frac{n}{b}) + f(n)$$

avec
$a \geq 1$, $b > 1$ et f asymptotiquement positive.

- "diviser pour régner" : division d'un problème en a sous-problème de taille $\frac{n}{b}$
- $T(\frac{n}{b}$ temps d'exécution d'un sous-problème
- $f(n)$ temps d'exécution de la division et de la fusion

Théorème

borne supérieure

$T(n) = aT(\frac{n}{b}) + f(n)$ avec $a \geq 1$, $b > 1$ et f asymptotiquement positive.
Posons $\beta = \log_b(a)$.
Alors :

1.
   - si $\exists\ \alpha > 0$ tel que $f(n) = \mathcal{O}(n^{\beta-\alpha})$
   - alors $T(n) = \Theta(n^{\beta})$
2.
   - si $f(n) = \Theta(n^{\beta})$
   - alors $T(n) = \Theta(n^{\beta} log(n))$
3.
   - si $n^{\beta-\alpha} = \mathcal{O}(f(n))$ et $\exists\ c < 1$ tel que $af(\frac{n}{b}) \leq cf(n)$ pour suffisamment grand,
   - alors $T(n) = \Theta(f(n))$

Utilisation du théorème

Il faut donc comparer $f(n)$ avec n^β avec $\beta = \log_b(a)$.

- Cas 1 : $f(n)$ plus petit que n^β, $T(n) = \Theta(n^\beta)$
- Cas 2 : $f(n)$ équivalent à n^β, $T(n) = \Theta(n^\beta log(n))$
- Cas 3 : $f(n)$ plus grand que n^β et une condition de majoration de sur f, $T(n) = \Theta(f(n))$