Algorithmique et structures de données
CM 9 – complexité dans le pire des cas et en moyenne
classes de complexité

Jean-Marie Le Bars
jean-marie.lebars@unicaen.fr

Plan du CM 9

Classification des algorithmes

Complexité d'un algorithme

Classification selon la complexité

Plan du CM 9

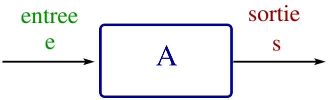
Classification des algorithmes

Complexité d'un algorithme

Classification selon la complexité

Classification des algorithmes

Schéma général d'un algorithme



- e appartient à un ensemble E qui est l'ensemble des entrées possibles

Vocabulaire

Les trois termes sont équivalents

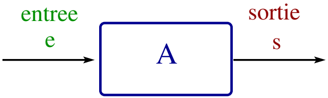
- entrée
- donnée
- instance

Classification

Il existe différents critères pour classifier les algorithmes.

Algorithmes déterministes

Schéma général d'un algorithme



Définition d'un algorithme déterministe

L'algorithme *A* est déterministe si, pour une entrée *e* fixée, il s'exécute toujours de la même façon.

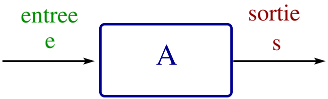
- même résultat : même sortie *s*
- même exécution : on effectue exactement les mêmes instructions

Remarque

- la plupart des algorithmes que l'on définit et utilise sont des algorithmes déterministes
- c'est le cas pour tous les algorithmes étudiés dans ce cours

Algorithmes probabilistes

Schéma général d'un algorithme

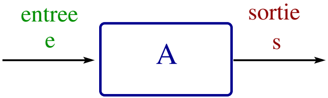


Définition d'un algorithme probabiliste

- l'algorithme A est un algorithme probabiliste lorsqu'il utilise des aléas
- un aléa est une tournure imprévisible que peut prendre un évènement
- exemple : on jette une pièce de monnaie pour savoir si on doit tourner à gauche ou à droite

Algorithme associé à un problème de décision

Schéma général d'un algorithme



s est un booléen : il prend la valeur VRAI ou FAUX.

Problème de décision

- le problème de décision correspond à une propriété que l'entrée *e* peut avoir ou ne pas avoir.
- la sortie est donc indépendante de l'algorithme et ne dépend que du problème de décision.
- les étapes pour atteindre le résultat peuvent être différentes d'un algorithme à un autre.

Exemple

- entrée $e = (T, x)$, où T est un tableau et x une valeur.
- sortie l'algorithme renvoie VRAI si x apparaît dans T et FAUX sinon.

Autres types de sortie

Même type d'entrée que de sortie

La sortie et l'entrée (ou une partie de l'entrée) peuvent appartenir au même ensemble.
Exemples

- l'entrée et la sortie sont des tableaux
  - exemple : tri d'un tableau
- l'entrée et la sortie sont des listes chaînées
  - exemple : construction de la liste inversée
- l'entrée et la sortie sont des arbres binaires
  - exemple : échange entre A_G et A_D

La sortie est calculée avec les données de l'entrée

- plus grand élément d'un tableau ou d'une liste chaînée
- somme des éléments d'un tableau ou d'une liste chaînée
- nombre de nœuds d'une liste chaînée ou d'un arbre, hauteur d'un arbre

Plan du CM 8

Classification des algorithmes

Complexité d'un algorithme

Classification selon la complexité

Complexité d'un algorithme

Complexité en temps

On s'intéresse au temps mis par l'algorithme A pour renvoyer la sortie s.

Complexité en mémoire ou en espace

On calcule la mémoire allouée pendant l'exécution de l'algorithme.

Calcul indépendant du langage de programmation et de la machine

On souhaite calculer la complexité en temps ou en espace sans tenir compte

- du choix du langage de programmation
- de l'environnement informatique : matériel informatique, système d'exploitation, ...

Nous nous intéresserons souvent d'abord à la complexité en temps.

Complexité en temps
Recherche d'un élément dans un tableau

Nous allons voir comment modéliser la complexité en temps d'un algorithme à partir d'un algorithme très simple, la recherche d'un élément dans un tableau

Algorithme A

```
Entrée e = (T : tableau d'entiers, taille : entier, x : entier)
Sortie s = i : entier, indice de la position de x dans T
ou -1 si x n'appartient pas à T

recherche(T : tableau, taille : entier, x : entier) : entier
   i = 0
   tant i < taille et T[i] <> x faire
      i = i + 1
   si i = taille alors retourner -1
   retourner i
```

Coût de l'algorithme

- Opération élémentaire : on choisit la comparaison entre $T[i]$ et x
- nous noterons $f(A, e)$ le coût de l'algorithme A sur l'entrée e
- $f(A, e)$ sera donc ici est le nombre de comparaisons entre $T[i]$ et x

Complexité dans le pire des cas
Recherche d'un élément dans un tableau

Coûts possibles

Posons n la taille du tableau T.

- nous avons $f(A, e) = i$ avec $i \in \{0, \dots, n-1\}$
- coût minimum $f_{min}(n) = 1$ l'élément x est en position 0
- coût maximum $f_{max}(n) = n$ l'élément x est en position $n - 1$ ou n'appartient pas au tableau.

La complexité dans le pire des cas est le coût maximal, elle vaut donc

$$f_{max}(n) = n.$$

Généralement, lorsque ce n'est pas spécifié, on considère la complexité dans le pire des cas lorsque l'on parle de complexité.

Complexité en moyenne
Recherche d'un élément dans un tableau

Distribution de probabilité sur les entrées

Il existe plusieurs façons de définir la complexité en moyenne.
Pour définir une complexité en moyenne, il faut définir une distribution sur les entrées.
Si $E = \{e_1, \ldots, e_n\}$ est l'ensemble des entrées.
On définit p_i, la probabilité que e_i soit l'entrée donnée pour l'exécution de notre algorithme.

Variable aléatoire

Le coût C est une variable aléatoire.

Calcul du coût moyen – espérance de C

Notons C_i le coût de l'algorithme sur l'entrée e_i.
C'est $E[C]$ l'espérance de C.

$$E[C] = \sum_{i=1}^{n} C_i\, p_i.$$

Complexité en moyenne
Recherche d'un élément dans un tableau

Entrées

Pour l'entrée, on ne tient compte que de la place de x dans le tableau.
$E = \{-1, 0, \ldots, n-1\}$.
-1 correspond au cas où l'élément x n'appartient pas au tableau.

Calcul des coûts

$$\begin{array}{lcl} C_i & = & i+1, \text{ pour } \in \{0, \ldots, n-1\} \\ C_{-1} & = & n \end{array}$$

Calcul de la moyenne

$$E[C] = \sum_{e_i \in E} p_i \, C_i.$$

Complexité en moyenne
Recherche d'un élément dans un tableau

1) Distribution uniforme sur les cases du tableau

- on suppose que x appartient au tableau donc $p_{-1} = 0$
- et qu'il a la même probabilité de se trouver à toute case du tableau
  - distribution uniforme ou équiprobabilité ou équirépartition

Distribution de probabilité

$$p_i = \frac{1}{n}, \text{ pour tout } i \in \{0, \ldots, n-1\}.$$

Espérance

$$\begin{aligned} E[C] &= \sum_{i=1}^{n} i \Pr(C = i) \\ &= \frac{1}{n}(1 + 2 + \ldots + n) \\ &= \frac{1}{n} \frac{n(n+1)}{2} \\ &= \frac{n+1}{2} \end{aligned}$$

Complexité en moyenne
Recherche d'un élément dans un tableau

2)Seconde distribution de probabilités

- x a une chance sur deux de ne pas appartenir au tableau
- pour les autres entrées, nous avons la distribution uniforme sur les indices

Calculs des probabilités

$$\begin{aligned} p_{-1} &= \frac{1}{2} \\ p_i &= \frac{1}{2n}, \text{ pour tout } i \in \{0, \ldots, n-1\}. \end{aligned}$$

Espérance

$$\begin{aligned} E[C] &= n\,\frac{1}{2} + \sum_{i=0}^{n-1}(i+1)\frac{1}{2n} \\ &= \frac{n}{2} + \frac{n+1}{4} \\ &= \frac{3n+1}{4} \end{aligned}$$

Complexité en moyenne
Recherche d'un élément dans un tableau

3) Troisième distribution de probabilités

- on suppose que x appartient au tableau donc $p_{-1} = 0$
- p_i croît lorsque i augmente

Choix des probabilités

$$p_i = \frac{2i}{n(n+1)}.$$

On vériie que l'on a $p_0 + p_1 + \ldots + p_{n-1} = 1$.

Calcul de l'espérance

$$\begin{array}{rcl} E[C] & = & \sum_{i=0}^{n-1} p_i C_i \\ & = & \frac{2}{n(n+1)}(1*1+2*2\ldots+n*n) \\ & = & \frac{2}{n(n+1)}\frac{n(n+1)(2n+1)}{6} \\ & = & \frac{2n+1}{3} \end{array}$$

Plan du CM 8

Classification des algorithmes

Complexité d'un algorithme

Classification selon la complexité

Classification selon la complexité

Objectifs

- le calcul du coût n'est pas toujours exact (c'est généralement difficile ou impossible)
  - on choisit les opérations que l'on va compter

Comparaison entre deux algorithmes

- on veut pouvoir comparer deux algorithmes résolvant le même problème
  - quel est le meilleur algorithme ?

Comparaison entre deux problèmes

- on veut pouvoir comparer deux algorithmes résolvant deux problèmes différents
  - quel est le problème le plus difficile ?

Classification selon la complexité

Méthode de classification

On décide de regrouper les algorithmes par classe.

Algorithmes en temps constant

Le coût de l'algorithme ne dépend pas de la taille de la donnée.

- accès à un élément d'un tableau
- insertion au début dans une liste chaînée

Algorithmes en temps linéaire

Le coût de l'algorithme vaut plusieurs fois la taille de la donnée.
Par exemple, il nécessite de parcourir une ou plusieurs fois l'entrée.

- calcul du nombre de nœuds d'une liste chaînée
- calcul du plus grand élément d'un tableau
- tri pour le drapeau hollandais

Comparaison asymptotique – O (grand o)

Définition mathématique

Soient $f : \mathbb{N} \rightarrow \mathbb{N}$ et $g : \mathbb{N} \rightarrow \mathbb{N}$.
On note $f(n) = O(g(n))$ lorsque

$$\exists k \in \mathbb{N}\ \exists N \in \mathbb{N}\ \forall n \geq N\ \ f(n) \leq k\ g(n).$$

Exemple

Prenons les deux fonctions f et g

- $f(n) = 4n + 15$
- $g(n) = 3n$

La majoration de f par g doit être vrai à partir d'un certain entier N.

Ici avec $k = 2$ et $N = 8$, nous avons bien

$$f(n) \leq 2g(n), \text{ pour tout } n \geq 8.$$

Nous pouvons donc écrire $f(n) = O(g(n))$.

Comparaison asymptotique – O (grand o)

Application à la complexité

$f(n)$ pourra représenter

- le coût d'un algorithme A sur une entrée e de taille n
- la complexité de A dans le pire des cas pour des entrées de taille n
- la complexité en moyenne de A pour une distribution de probabilité sur les entrées de taille n

Remarque : la notation est asymptotique

- l'algorithme doit pouvoir s'appliquer sur des entrées de taille aussi grande que l'on souhaite
- pour des entrées de taille limitée la notation n'a pas de sens, elle ne doit pas être utilisée

Notation O pour la majoration

Majoration du coût

Pour majorer le coût la notation O est très utile. Par exemple, si $f(n)$ est le coût de la recherche d'un élément dans un tableau, nous avons, quelle que soit l'entrée e,

$$f(n) = O(n).$$

Une notation imprécise

Si $f(n) = O(g_1(n))$ et $g_1(n) \leq g_2(n)$ alors $f(n) = O(g_2(n))$.
Par exemple, pour le coût de la recherche d'un élément dans un tableau, nous avons

$$f(n) = O(n),$$

mais nous avons aussi les majorations suivantes

$$\begin{array}{lcl} f(n) & = & O(n^2) \\ f(n) & = & O(n^4) \\ f(n) & = & O(2^n) \end{array}$$

On voit bien que dire dans le cas de la recherche que l'on a $f(n) = O(2^n)$ n'est pas très informatif !

Comparaison asymptotique – Θ (grand theta)

Définition

Soient $f : \mathbb{N} \to \mathbb{N}$ et $g : \mathbb{N} \to \mathbb{N}$.
On note $f(n) = \Theta(g(n))$ lorsque

$$f(n) = O(g(n)) \text{ et } g(n) = O(g(n)).$$

Cela revient à dire que l'on a

$$\exists k_1 \in \mathbb{N}\ \exists k_2 \in \mathbb{N}\ \exists N \in \mathbb{N}\ \forall n \geq N \quad k_1 g(n) \leq f(n) \leq k_2\, g(n).$$

Relation d'équivalence

On définit la relation binaire $\mathcal{R}$ sur les fonctions de $\mathbb{N} \to \mathbb{N}$ par

$$f \quad R\ g \text{ lorsque } f(n) = \Theta(g(n)).$$

$\mathcal{R}$ est une relation d'équivalence

- réflexivité $f(n) = \Theta(f(n))$
- symétrie si $f(n) = \Theta(g(n))$ alors $g(n) = \Theta(f(n))$
- transitivité si $f(n) = \Theta(g(n))$ et $g(n) = \Theta(h(n))$ alors $f(n) = \Theta(h(n))$

Θ – Terme dominant

La classe d'une fonction $f(n)$ dépend de son terme dominant.

Classe linéaire

Les fonctions suivantes appartiennent toutes à la classe linéaire :

- $f(n) = 2n + 10$
- $f(n) = \dfrac{n}{3}$
- $f(n) = n + 3\ln n + 7$
- $f(n) = 3n + \sqrt{n} - 3\ln n$

Classe quadratique

Les fonctions suivantes appartiennent toutes à la classe quadratique :

- $f(n) = n^2 - 10n$
- $f(n) = 2n^2 + \frac{n}{3}$
- $f(n) = 3n^2 + 4n\ln n + 3n + 6\ln n$
- $f(n) = n^2 + n^{\frac{3}{4}}$

Complexité d'un problème $\neq$ complexité d'un algorithme

Complexité d'un problème

La complexité d'un problème est une fonction $f(n)$ telle que pour tout algorithme A résolvant ce problème et $g(n)$ la complexité de cet algorithme, nous avons

$$f(n) \leq g(n).$$

Observations

- $f(n)$ n'est pas toujours connu
- on ne sait pas toujours s'il n'existe pas un meilleur algorithme (de complexité plus petite) que les algorithmes connus
- la définition s'applique aussi bien pour la complexité dans le pire des cas et en moyenne

Complexité d'un problème $\neq$ complexité d'un algorithme

Complexité d'un problème

La complexité d'un problème est la fonction $f(n)$ lorsque, pour tout algorithme A résolvant ce problème et $g(n)$ la complexité de cet algorithme, nous avons

$$f(n) \leq g(n).$$

Complexité d'un algorithme

La complexité d'un algorithme A résolvant un problème donne une majoration de la complexité du problème.

- si $g(n)$ est la complexité de A alors $f(n) = O(g(n))$
- si la complexité de A est en $O(g(n))$ alors $f(n) = O(g(n))$

Ordre de complexité des algorithmes

$ns = 10^{-9}s$, nanoseconde, $\mu s = 10^{-6}s$, micro seconde.

Ordres de grandeur et taille des entrées

ordre de grandeur	nom de la classe	exemple d'algorithme	$n = 10$	$n = 50$	$n = 250$	$n = 10^3$	$n = 10^4$	$n = 10^5$
$\Theta(1)$	constant	accès dans un tableau	$10ns$	$10ns$	$10ns$	$10ns$	$10ns$	$10ns$
$\Theta(\log n)$	logarithmique	recherche dichotomique	$10ns$	$20ns$	$30ns$	$30ns$	$40ns$	$60ns$
$\Theta(n)$	linéaire	parcours d'une liste	$100ns$	$500ns$	$2.5\mu s$	$10\mu s$	$100\mu s$	$10ms$
$\theta(n \log n)$	quasi-linéaire	tri rapide	$100ns$	$850ns$	$6\mu s$	$30\mu s$	$400\mu s$	$60ms$
$\Theta(n^2)$	quadratique	tri lent	$1\mu s$	$25\mu s$	$625\mu s$	$10ms$	$1s$	$2.8h$
$\Theta(n^3)$	cubique	multiplication matricielle naïve	$10\mu s$	$1.25ms$	$156ms$	$10s$	$2.7h$	316 ans
$\Theta(n^k)$	polynomial	test de primalité						
$\Theta(2^n)$	exponentiel	SAT	$10\mu s$	130 jours	10^{59} ans			