BUT Informatique - R3.02
Conception d'algorithmes corrects et efficaces
Algorithmes corrects et efficaces

J. Christian Attiogbé

Novembre 2023

Nantes Université

Motivations

A travers cet enseignement, vous allez

- acquérir les connaissances et compétences de base pour analyser, concevoir et construire des algorithmes corrects,
- mettre en œuvre les compétences acquises sur des exemples et études de cas.
- acquérir les compétences prévues dans la R3.02 :
  - **AC21.03 :** Adopter de bonnes pratiques de conception et de programmation
  - **AC22.01 :** Choisir des structures de données complexes adaptées au problème
  - **AC22.02 :** Utiliser des techniques algorithmiques adaptées pour des problèmes complexes (par ex. recherche opérationnelle, méthodes arborescentes, optimisation globale, intelligence artificielle...)
  - **AC22.04 :** Évaluer l'impact environnemental et sociétal des solutions proposées

Organisation

Moyens

Formation encadrée : CM 6h (4*1h20) - TD/TP 12h40 (5 sem., 2*1h20/sem)
Prérequis : InitDev R1.01 (BUT1)
Fil conducteur : Exercices de complexité graduelle à réaliser

Intervenants

Christian Attiogbé (CM, TD-TP)
Jean-François Berdjugin (TD-TP)

Plan de la suite (adaptation selon contraintes)

Conception d'algorithmes corrects

Motivation

Exercice

Considérons un programme qui effectue **un traitement (élémentaire) donné d'un dossier en 1**μs (par exemple par la sécurité sociale).

Admettons qu'un dossier corresponde à une personne, mais peut en impliquer d'autres.
La population française est estimée à **67 500 000** de personnes.

Combien de temps mettra le programme pour traiter tous les dossiers ?

Motivation

Exercice

Considérons un programme qui effectue **un traitement (élémentaire) donné d'un dossier en 1**μs (par exemple par la sécurité sociale).

Admettons qu'un dossier corresponde à une personne, mais peut en impliquer d'autres.
La population française est estimée à **67 500 000** de personnes.

Combien de temps mettra le programme pour traiter tous les dossiers ?
Si le programme est en **complexité de** n^2, il faudrait environ 1890 heures, soit environ 79 jours pour que le traitement soit fini.

Motivation

Exercice

Considérons un programme qui effectue **un traitement (élémentaire) donné d'un dossier en 1**μs (par exemple par la sécurité sociale).

Admettons qu'un dossier corresponde à une personne, mais peut en impliquer d'autres.
La population française est estimée à **67 500 000** de personnes.

Combien de temps mettra le programme pour traiter tous les dossiers ?
Si le programme est en **complexité de** n^2, il faudrait environ 1890 heures, soit environ 79 jours pour que le traitement soit fini.
Le même traitement avec un programme en **complexité linéaire de** n, prendrait un peu plus d'une minutes !

Motivation

Hypothèse : pour une instruction/opération s'exécutant en 1μs ($10^{-6}s$).

	complexité				
taille ↓	$log_2(n)$	n	$nlogn$	n^2	2^n
n=10	3μs	10μs	30μs	100μs	1000μs
n=100	7μs	100μs	700μs	1/100s	10^{14} siècles
n=1000	10μs	1000μs	1/100 s	1s	astronomique*
n=10000	13μs	1/100 s	1/7 s	1,7 min	astronomique
n=100000	17μs	1/10 s	2 s	2,8h	astronomique

Table: Temps d'exécution en fonction des coûts et taille des données

(*) astronomique = 1 milliard de milliard d'années ! (10^{18} années)
Un programme en n^2 mettra 2,8h pour une donnée de taille 100 000

Introduction

- L'activité de construction de logiciels (appelée aussi *développement* de logiciels) nécessite des **méthodes rigoureuses** pour assurer de **bonnes propriétés** aux logiciels produits.
- Étapes de construction de logiciels : analyse, conception, développement, tests, maintenance.
- Cet enseignement concerne plus l'étape de **conception**, charnière entre l'analyse et le développement ; elle en conditionne les résultats.

☞ **conception d'algorithmes corrects et efficaces**, au service de la construction rigoureuse (efficace ?) des logiciels.

Introduction

Algorithmes, à la base des *logiciels* (applications logicielles) et des systèmes intégrant des logiciels.

La **correction, la qualité, la performance, le coût**, etc des applications dépendent en partie de leur bonne conception et à un certain niveau de précision, des algorithmes utilisés dans différentes parties des logiciels.

Ces algorithmes, comme les logiciels engendrés ont des **coûts calculables, mesurables ou estimables**.

Introduction

Les coûts permettent également de **comparer les algorithmes/logiciels en terme de performance**,...

En fonction des critères choisis par le programmeur, on peut **calculer ou estimer** le coût d'un programme/algorithme.

Généralement,
le coût en **temps**, le coût en **mémoire**
(et impact sur l'environnement : **énergie consommée par les processeurs, le réseau**),
on peut quantifier d'**autres coûts** selon les projets.

Introduction

Pour évaluer les coûts, on peut **compter** :

- les **opérations élémentaires :** affectation de valeur à une variable, comparaison d'éléments, ...
- les **opérations non élémentaires :** les accès à la mémoire, les entrées/sorties vers des fichiers ou périphériques, les accès réseau, etc

A partir des temps consommés par les opérations, on estime/calcule le coût en temps de l'algorithme/programme en fonction de la quantité de ces opérations effectuées.

Introduction

De la même façon que pour les opérations, à partir des **structures de données utilisées**, on estime/calcule l'espace mémoire occupé ou utilisé par un algorithme/programme.

Par exemple, les structures de données statiques (Tableaux) et dynamiques (listes, arbres, graphes) n'engendrent pas les mêmes coûts, et ne satisfont pas les mêmes besoins pour les programmeurs.

Pour estimer et analyser les coûts, on utilise les **ordres de grandeur**.

Focus de l'enseignement

Nous étudierons dans la suite du cours

- les outils d'analyse des algorithmes et
- des méthodes de construction correcte des algorithmes et logiciels.

Algorithme et problème

Un algorithme résout un problème.

Algorithme

Un algorithme est une **méthode de résolution systématique ou de traitement systématique d'un problème** donné ou de réalisation d'**un calcul**, par une suite d'actions organisées par des structures dites de contrôle, en s'appuyant éventuellement sur des **données d'entrée**.

Il y a des problèmes qui n'ont pas de solution algorithmique.
Exemple : problème des chemins de Euler : pas de solution !

Il y a un risque à ne pas savoir identifier les natures des problèmes qu'on a à traiter.
☞ Plusieurs méthodes, techniques, outils, disciplines existent.

Analyse de la complexité

L'**analyse de la complexité** fournit des outils et des mesures pour comparer des algorithmes du point de vue de leur efficacité ou performance.

La **théorie de la complexité** elle, s'attache elle à la **calculabilité**, et l'étude des classes de problèmes, **solubles ou non solubles**...

Calculabilité - (Théorie de la complexité)

L'étude de la calculabilité permet d'identifier des classes de problèmes ou d'objets mathématiques pour lesquels il existe des solutions algorithmiques. En effet tous les problèmes ne peuvent pas être résolus algorithmiquement.

Ordres de grandeur

On compare les algorithmes (et aussi les programmes dérivés), en calculant les **quantités** de ressources **temps** ou d'**espaces mémoires** utilisées. Ces quantités considérées sont les ordres de grandeur (au lieu des valeurs exactes). On les **estime** en prenant en compte par exemple les cas extrêmes (favorable, défavorable, pratique).

Formellement le coût d'un algorithme est donné par une fonction $f(n)$.

Avec les ordres de grandeur on **majore**, ou on **encadre** $f(n)$ par des fonctions dont les comportements/croissances sont connues.

Par exemple, le coût d'un algorithme en fonction de la taille n des données, peut être donné par une fonction $f(n) = n(n+1)/2$

Ordres de grandeur

Ordres de grandeur

Un ordre de grandeur est un ensemble de fonctions (ou une classe), utilisé comme référence pour apprécier la croissance d'une fonction de coût : n^2, $log_2(n)$, n^p, etc

$$\mathcal{O}(g(n)) = \{\ h \in \mathbb{N} \to \mathbb{R}_+ \mid \exists(c, n_0).(c \in \mathbb{R}, n_0 \in \mathbb{R} \quad \wedge$$
$$\forall n.((n \in \mathbb{N} \ \wedge \ n \geq n_0) \ \Rightarrow \ h(n) \leq c.g(n))) \ \}$$

Ainsi, on exprime que le coût $f(n)$ d'un algorithme est dans $\mathcal{O}(g(n))$

Quelques ordres de grandeur

Classe	dénomination ou type de complexité
$O(n!)$	complexité factorielle ; démentiel
$O(2^n)$	complexité exponentielle ; coût catastrophique quand taille n des données grandit.
$O(n^2)$	complexité quadratique ; acceptables pour nombreux algorithmes pour des tailles assez grandes des données (> 10-100taines milliers).
$O(nlogn)$	**complexité pseudo-linéaire**.
$O(n)$	**complexité linéaire** (liée à la taille n des données),
$O(logn)$	**complexité logarithmique** (en base 2).
$O(1)$	en temps constant, indépendamment des entrées de l'algo.

Outils de comparaison des algorithmes.
Performance et **efficacité** sont liées à la **classe de complexité**.

Terminologie

- complexité spatiale : coût en quantité d'espace mémoire
- complexité temporelle : coût en rapidité d'exécution
- **grand O** (O) : comportement asymptotique (taille des données très grande, aux abords de l'infini).
  O donne un ensemble de majorants d'une fonction.
  thêta (Θ) : comportement dans les cas pratiques ;
  Θ donne une mesure plus juste intégrant minorants et majorants.
  petit o (o) : comportement dans les cas favorables ; donne les minorants.

Pour aller plus loin

de nombreux ouvrages disponibles :
- Introduction à l'algorithmique de Cormen, Leiserson, Rivest
- Mathématiques concrètes de Graham, Knuth, Patashnik
- Mathématiques discrètes et informatique, de N.H. Xuong

et des liens d'entrée sur de nombreuses ressources pédagogiques :
- `https://fr.wikipedia.org/wiki/Complexit%C3%A9_en_temps`
- `https://en.wikipedia.org/wiki/Time_complexity#Table_of_common_time_complexities`