I21 : Algorithmique

Axel Coezard

Janvier 2020

Table des matières

Chapitre 1

Algorithmique

1.0 Introduction

n. b. L'objet de ce cours est l'étude des algorithmes.

def. Un algorithme est une procédure permettant de résoudre un problème en un nombre fini d'étapes.

1.1 Problème et instance d'un problème

1.1.1

def. Un problème algorithmique est défini par :
- son entrée
- sa sortie

e. g. Le tri
- entrée : ensemble de n ¿= 1 données a_1, a_2, ..., a_n.
- sortie :

IL est important d'être très précis dans les specifications du problème. Une définition trop vague peut rendre le problème impossible à résoudre.

e. g. Meilleur chemin
- entrée : une carte routière et 2 points A et B.
- sortie : le meilleur chelin entre A et B.

n. b. Ici la notion de meilleur chemin est trop vague. Il peut s'agir du plus court chemin, du plus rapide, du moins cher, etc.

def. On appelle instance d'un problème la donnée d'une entrée spécifique du problème. Pour être validé un algo doit résoudre le problème pour n'importe laquelle de ses instance.

e. g. : Problème de tri
- {235, 104, 15, -7, 0}
- {Nicolas, Valérie, JP, Joseph}

n. b. En fonction du type de données on peut avoir différentes opérations acceptables.

1.1.2 Description d'un algorithme

def. On utilise 3 types de langages pour écrire un algorithme :
- Le langage naturel
- Le pseudocode
- Le langage de programmation

e. g. Somme de n premiers nombres
- entrée : un entier n
- sortie : $\sum_{i=1}^{n} i = 1 + 2 + ... + n$

1.1.3 Pseudocode

def. Il s'agit d'un langage de universel de description algorithmique. Celui-ci ne depend ni d'un langage ni de l'architecture de la machine censé faire tourner le code.

e. g. Dans ce cas, un algorithme est composé de 3 blocs :
- Les données
- Les variables
- Les instructions

n. b. Les Mots-Clef : DEBUT, FIN, SI, ALORS, SINON, TQ, FTQ, FAIRE

n. b. L'affectation se fait avec le symbole $\Leftarrow$

n. b. Utilise les opérations mathématiques standards.

n. b. Les tableaux sont de tailles fixes. Ils commencent à l'indice 1. On accède à un élement via [ ].

1.1.4 Justesse et terminaison

def. Un algorithme pour être considéré valide doit satisfaire 2 critères et doit :
- Être juste / correct.
- Se terminer.

n. b. Noter la justesse ou la terminaison d'un algorithme se fait avec une preuve mathématique.

def. Les preuves de justesses sont en géneral très dures, et ne sont pas effectuées.

def. Les preuves de terminaison sont beaucoup plus simple et permettront de calculer la complexité des algorithme.

n. b. Ces données doivent permettre de garantir qu'un algorithme se termine quel quelle que soit l'instance du problème.

1.1.5 Preuve par récurrence

def. La preuve par récurrence est l'autre preuve principale des preuves d'arrêt. Généralement, elles nous serviront à prouver des formules de la forme :

$$\sum_{i=1}^{n} u_i = F(n)$$

n. b. Le but d'une telle preuve est de prouver la validité d'une propriété P(n) pour tous les entiers supérieur à une borne. Elle se déroule en deux étapes :
- L'initialisation.
- L'hérédité.

1.1.6 Invalidation de l'algorithme

n. b. Autant il peut être difficile de prouver la justesse d'un algorithme, autant il est très simple de prouver qu'un algorithme est faux.

i. e. Il suffit de trouver une instance du problème à résoudre pour laquelle l'algorithme renvoie un résultat faux.

e. g. Ramassage de Plots
- entrée : n plots sur un terrain
- sortie : parcours permettant de ramasser tous les plots et de revenir au départ.

Chapitre 2

Analyse des algorithmes

2.1

def. Analyser un algorithme c'est prévoir la quantité de ressources nécessaire à son exécution. Les ressources peuvent être :

- Le temps.
- La mémoire.
- La consommation électrique.
- Le coût financier.

n. b. Dans ce cours, nous nous intéressons essentiellement à la dimension temporelle.

def. Une analyse hyper détaillée consiste à pondérer à chaque opérations et à compter leur nombre d'occurences (en fonction de n).

- C_A : côut de l'addition
- C_m : côut de la multiplication
- C_d : côut de la division
- C_f : côut de l'affectation
- C_p : côut de la comparaison

n. b. Pour réaliser des analyses qui soient les plus générales possibles, **i. b.** qui ne dépendent pas de l'architecture de la machine cible, on considère un modèle de machine abstraitre : le modèle **R**andom **A**ccess **M**emory

e. g. Machine hypothétique pour laquelle :

- Les opérations simples ($+$, -, $\times$, $\div$, $\leq$, $=$, $\Leftarrow$) consommnent une unité de temps.
- Les boucles sont des compositions d'opérations simples, leur coût dépend du nombre d'opération multiplié par le nombre d'itérations.
- Un accès mémoire consomme une unité de temps.

— La quantité de mémoire n'est pas limitée.

2.2 Analyse asymptotique

n. b. A priori, plus une donnée en entrée est grande, plus un algorithme de résolution est lent. Pour étudier l'efficacité d'un algorithme, on considère toujours une donnée à taille fixée.

n. b. L'analyse asymptotique nous renseigne sur la manière dont évolue le coût de calcul quand la taille des données augmente.

e. g. Recherche d'un élement dans un tableau :
- n cases à tester.
- 5 cases : 5 cases à tester.
- 10 cases : 10 cases à tester.

n. b. Même à taille fixée, certaines instance peuvent être plus faciles à résoudre que d'autres.

e. g. Recherche d'un élément :
- Cas facile : l'élément recherché est le premier élément du tableau (1 tour de boucle).
- Cas difficile : l'élément dans la dernière case (m tours de boucles).

n. b. On considère du coup 3 types de notations pour évaluer un algorithme :
- $\mathrm{T}(n)$: le coût de calcul (la complexité générale).
- $T^{\vee}(n)$: la complexité en meilleur (facile).
- $T^{\wedge}(n)$: la complexité en pire (difficile).

n. b. Dans le meilleur cas, l'algorithme effectue un nombre normal d'étapes de calcul pour n'importe quelle instance de taille n. Dans le pire cas, l'algorithme en effectue un nombre maximal.

2.3 Notation asymptotique

def. La complexité d'un algorithme est toujours une fonction numérique, elles sont en général difficiles à manipuler.
- Trop de as à gérer (il n'existe par une seule formule proche).
- Trop complexe l'écriture fait intervenir de nombreux termers qui n'apportent aucune information.

n. b. Pour se simplifier l'analyse, on utilise les notations asymptotiques suivantes : f et g sont des fonctions numériques positives.

def. f(n) = **O** (g(n))

i. e. Quand n devient grand, g(n) est toujours plus grand que f(n).

def. f(n) = Ω (g(n))

i. e. Quand n devient grand, g(n) est toujours plus petit que f(n).

def. f(n) = Θ (g(n))

i. e. Quand n devient grand, g(n) croit aussi rapidement que f(n).

prop. On dispose des règles de calcul suivantes :
- f(n) + g(n) = **O** (max(f(n),g(n)))
- c × f(n) = **O** (f(n))
- f(n) × g(n) = **O** (f(n) × g(n))

n. b. Valable également pour Ω et Θ.

n. b. Lorsqu'une analyse asymptotique fourni une complexité dans le pire cas différente de celle dans le meilleur cas, la complexité générale est en grand **O** du pire cas.

2.4 Familles de complexité

n. b. Dans la suite de ce cours, toutes les fonctions considérés sont positives et croissantes. Dans ce cadre, pour évaluer le rapport entre les comportements asymptotiques de 2 fonctions f et g il suffit d'étudier : $\lim_{n\to+\infty} \frac{f(n)}{g(n)}$
- Si l = $+\infty$, alors on a f(n) = Ω (g(n)).
- Si l = 0, on a g(n) = Ω (f(n)).
- Si l $\in \mathbf{R}_+^*$, on a f(n) = Θ (g(n)).

2.4.1 Logarithmique

def. Le Logarithme s'écrit log(n).

def. Le Logarithme de base a s'écrit $log_a(n) = \frac{ln(n)}{ln(a)}$.

n. b. En particulier, log_2 est le plus utilisé en information et on le note parfois log ou lg.

n. b. $b^{log_b(a)} = a$ et $2^{log_2(n)} = n$

def. $log_a(b \times c) = log_a(b) + log_a(c)$

2.4.2 Polynomiale

def. Tout polynome s'écrit $f(n) = a_k u^k + a_{k-1} u^{k-1} + ... + a_i u^i + a_0 u^0$

n. b. Propriétés
- f(n) = Θ (u^k).
- k = 1, on parle de complexité linéaire.
- k = 2, on parle de complexité quadratique.

n. b. On peut étendre la famille aux puissances réelles : $u\sqrt{u} = u^{1.5}$.

2.4.3 Exponentielle

Chapitre 3

Bases d'algorithmique

n. b. Dans la pratique, écrire un algo de résolution d'un problème donné revient à parcourir les données du problème dans un certain ordre. Dans ce cours, nous allons nous intéresser essentiellement au parcours de tableaux en 1 et 2 dimensions. Ces algorithmes de parcours sont un peu l'équivalent des gammes en musique.

3.1 Parcours d'un tableau de dimension 1

3.1.1 Parcours alterné

def. Le parcours alterné conciste à afficher le premier élement puis le dernier, le deuxième puis l'avant dernier, etc. Cet algorithme est par exemple utilisé pour résoudre le problème de détection des palimdromes.

e. g. Soit le tableau $T = [1, 2, 3, 4, 5]$, l'agorithme affiche : 1 5 2 4 3

algo.

n. b. La boucle dépend des variables i et j. i est initialisé à 1 et modifier à l'instruction $i \Leftarrow i + 1$. j est initialisé à n et modifier à l'instruction $j \Leftarrow j - 1$. Les valeurs de i sont celles de la suite : $u_0 = 1, u_{r-1} = u_r + 1$. i.e. $u_r = r + 1$. Les valeurs de j sont celles de la suite : $v_0 = n, v_{r+1} = v_r - 1$. i.e. $v_r = n - r$. La condition d'arrêt de la boucle ($i \gtrneqq j$) est satisfaite quand $u_r \gtrneqq u_r$.

n. b. On en déduit que le nombre de tour de boucle est $\frac{n-1}{2} + 1$. Toutes les opérations à l'interieur sont de complexité $\Theta(1)$. Donc $C(n) = \sum_{r=1}^{\frac{n-1}{2}+1} \Theta(1) = \Theta(n)$.

3.1.2 Parcours d'un tableau circulaire

algo.

n. b. Ici, un tableau circulaire est une représentation d'un ensemble de données sans début ni fin.

n. b. Pour résoudre le problème il faut être en mesure d'effectuer un parcours décalé circulaire du tableau.

algo.

n. b. Géneralement, un opérateur permettant de simplifier à la fois la conception et l'écriture des algorithmes de parcours de tableaux est le modulo : % ou *mod*.

algo.

n. b. Pour résoudre le problème initial on peut appliquer deux stratégies :
- Recherche l'indice du plus petit élément, puis faire un parcours circulaire du tableau à partir de cet indice et vérifier que tous les élements sont ordonnés.
- Un tableau circulaire est ordonnée si au plus une paire d'élements contigus sont mal ordonnés. La stratégie consiste alors à compter le nombre de ces paires.

algo.

3.2 Parcours de tableau à 2 dimensions

def. Formellement, un tableau à 2 dimensions est ce qu'on appelle une matrice. En algorithmique on représente cette structure par un tableau de tableau. Chaque sous-tableau représente une ligne.

3.2.1 Parcours en ligne

A FAIRE

3.2.2 Parcours en colonne

n. b. On reprend l'algorithme de parcours en ligne et on inverse la boucle de i et de j.

3.2.3 Parcours en serpentin

n. b. On applique le même algorithme que pour les lignes, mais à chaque lecture de colone, on incrémente ou décrémente en fonction de la parité de la ligne.

3.2.4 Parcous en diagonale

A FAIRE

3.2.5 Vérification triangulaire sup

A FAIRE

3.3 Algorithmes de tri

def. Il s'agit du problème le plus étudié en informatique. Il consiste à réorganiser des données en vue de faciliter leur traitement.

e. g. On concidère un tableau de taille n contenant des entiers : quel est le nombre qui apparait le plus ?

3.3.1 Idée naïve

n. b. Pour chaque élément du tableau on compte son nombre d'occurences.