Fondements de l'informatique
Logique, modèles, et calculs

Chapitre: Introduction

Cours CSC_41012_EP

de l'Ecole Polytechnique

Olivier Bournez

bournez@lix.polytechnique.fr

Version du 16 juillet 2024

Introduction

Ce cours est un cours sur les fondements de l'informatique : il se focalise sur trois domaines centraux en informatique : la logique, les modèles de calculs et la complexité.

Tous ces domaines sont reliés par la question suivante : quelles sont les capacités et les limites des ordinateurs ?

Même un téléphone est maintenant capable de résoudre très rapidement certains problèmes, comme trier un répertoire de plus d'un million d'entrées. Par contre, certains problèmes s'avèrent beaucoup plus lents et difficiles à résoudre : résoudre un problème d'emploi du temps, ou affecter les choix d'affectations des élèves de l'École Polytechnique en fonction de leurs préférences ordonnées est par exemple beaucoup plus difficile.

Au cœur de ce cours est la compréhension de ce qui fait qu'un problème, comme le tri, est simple à résoudre informatiquement, alors qu'un problème comme un problème d'emploi du temps, peut prendre des siècles à résoudre avec seulement un millier de données en entrée.

Autrement dit, au cœur de nos interrogations est aussi la question suivante : qu'est-ce qui rend certains problèmes difficiles, et d'autres faciles ?

C'est la question centrale de la théorie de la complexité, et de la calculabilité.

Pourquoi s'intéresser à comprendre les problèmes difficiles, plutôt que d'essayer de résoudre des problèmes très concrets ?

Premièrement, parce que des problèmes très simples et concrets, et aux enjeux économiques considérables, s'avèrent faire partie des problèmes difficiles.

Deuxièmement, parce que comprendre qu'un problème ne peut pas être résolu facilement est utile. Cela signifie que le problème doit être simplifié ou modifié pour pouvoir être résolu. Ce cours permet réellement de comprendre les pistes pour éviter les problèmes difficiles à résoudre informatiquement.

Enfin et surtout parce que les problèmes difficiles ont réellement des implications dans la conception de nombreux systèmes actuels. Par exemple, pour la vérification, l'analyse, et la conception de systèmes : lorsqu'on conçoit un système, on souhaite en général qu'il se comporte au minimum selon la spécification avec laquelle on l'a conçu. On aimerait que le processus de vérification puisse s'automatiser, c'est-à-dire que l'on puisse garantir informatiquement qu'un système donné vérifie une/sa spécification. Surtout, lorsque le système en question est d'une complexité énorme, comme les processeurs actuels, et qu'un unique être humain n'est plus capable d'en comprendre seul tous les composants.

Les résultats de ce cours montrent précisément que le processus de vérification ne peut pas s'automatiser facilement. Tout l'art de la vérification de systèmes, et donc de la conception de systèmes, est de tenter d'éviter ces difficultés pour la rendre praticable, ce qui nécessite d'avoir compris ces difficultés.

D'autres domaines sont fortement impactés par la complexité. Un des premiers qui l'a été historiquement est la cryptographie : dans la plupart des domaines, on cherche plutôt à privilégier les problèmes faciles à résoudre aux problèmes difficiles. La cryptographie est originale de ce point de vue, car elle recherche plutôt les problèmes difficiles à résoudre que les problèmes simples, car un code secret doit être dur à casser sans la clé secrète.

On peut aussi se demander : pourquoi autant de logique dans un cours sur les fondements de l'informatique ?

Une première raison est parce que les programmes informatiques et les langages informatiques sont essentiellement basés sur la logique. Les processeurs sont d'ailleurs essentiellement composés de portes logiques. Les programmes sont essentiellement faits d'instructions logiques, et de tests logiques. Comprendre cette logique permet de bien comprendre ce que font les programmes informatiques.

De façon fondamentale, les programmes et les systèmes informatiques obligent bien souvent à décrire très précisément les objets sur lesquels ils travaillent : pour résoudre un problème d'emploi du temps, le système va obliger à décrire toutes les contraintes. Pour faire une requête sur une base de données, le système va obliger à formuler très précisément cette requête. Il s'avère que la logique mathématique est un outil très naturel pour décrire le monde qui nous entoure, et à vrai dire, le modèle le plus naturel que nous connaissons pour le faire. Comprendre les concepts de la logique, permet de bien comprendre nombre de concepts informatiques. Par exemple, pour décrire le système d'information d'une entreprise, ou tout système complexe, le meilleur outil reste bien souvent la logique mathématique.

Une troisième raison est historique, et au cœur en fait de la naissance de l'informatique. Durant la première moitié du siècle dernier, des mathématiciens comme Kurt Gödel, Alonzo Church ou Alan Turing ont découvert que certains problèmes ne pouvaient pas se résoudre par des dispositifs informatiques ou automatiques comme les ordinateurs. Par exemple, le problème de déterminer si un énoncé mathématique est vrai ou non. Cette tâche, qui est le quotidien du mathématicien, ne peut pas être résolue par aucun ordinateur, quelle que soit sa puissance.

Les conséquences de ces résultats profonds ont permis la naissance d'idées sur des modèles d'ordinateurs qui ont mené à la conception des ordinateurs actuels.

Au cœur de ces découvertes sont des liens très forts qui unissent algorithmes et démonstrations : une démonstration logique correspond à un algorithme. A partir d'hypothèses, on déduit des nouvelles assertions à l'aide de règles logiques. Réciproquement, un programme correspond à une démonstration dans un certain sens.

Ce sont ces liens forts entre algorithmes et démonstrations qui ont fait naître l'informatique et ses concepts et ce bien avant l'existence même de machines aussi puissantes que celles que nous connaissons actuellement. Historiquement, la première question était : "qu'est-ce qu'une démonstration" ? Elle est maintenant devenue : "qu'est-ce qu'un ordinateur" ?

Ils ont par ailleurs révolutionné notre conception des mathématiques, de l'informatique, et plus généralement du monde qui nous entoure.

En mathématiques, ces liens ont mené à une crise des fondements, avec le retour sur des questions aussi fondamentales que celle-ci : qu'est-ce qu'un ensemble, qu'est-ce qu'une démonstration ? Que peut-on prouver ?

L'ambition de ce document est plutôt de se focaliser sur l'informatique. Nous estimerons que ce cours aura atteint son but si notre lecteur change au final la réponse qu'il aurait pu faire a priori sur des questions aussi simples que celle-ci :

— qu'est-ce qu'un ordinateur ?
— qu'est-ce qu'une preuve ?
— qu'est-ce qu'un algorithme ?
— qu'est-ce qu'un bon algorithme ?

Si tel est le cas, sa façon de programmer, ou d'appréhender un problème informatique ne devrait plus être la même.

Remerciements L'auteur de ce document souhaite remercier vivement Johanne Cohen, Bruno Salvy, David Monniaux pour leurs retours sur des versions préliminaires de ce document. Je remercie par ailleurs les promotions 2011-2012, 2012-2013, 2013-2014, 2014-2015, 2015-2016, 2016-2017, 2017-2018, 2018-2019, 2019-2020, 2020-2021, 2021-2022, 2022-2023, 2023-2024 pour leurs retours sur le cours, qui portait alors le code INF412. Des remerciements particuliers à Louis Abraham, Benjamin Allouche, Sariah Al Saati, Olivier Bailleux, Juliette Buet, Ismaël Cahu, Louis Cousturian, Xuanrui Fan, Carlo Ferrari, Léo Gaspard, Estienne Granet, Pierre-Jean Grenier, Alexis Le Dantec, Denis Langevin, Emmanuel Lazard, Stéphane Lengrand, Arnaud Lenoir, Elsa Lubek, Roberto Moura, Tom Niget, Maximilien Richer, Louis-François Rigano, Fabien Roger, Louis Rustenholz, David Savourey, André Sintzoff, Matthieu Vermeil, Noam Zeilberger et Zigfrid Zvezdin, pour des retours détaillés et des suggestions précises d'améliorations, ou pour avoir signalé des problèmes sur des versions antérieures de ce polycopié. Merci aussi à Romain Cosson et Rodrigue Lelotte pour des retours sur les corrections des sujets d'examen reliés.

Ce document doit aussi beaucoup à des retours faits par les étudiants du cours CSE304 de Bachelor à l'École Polytechnique sur un document en langue anglaise dont certains chapitres sont communs. Des remerciements spéciaux pour Louis de Benoist De Gentissart, Agathe De Vulpian, Guillaume Lainé and Skander Moalla pour leurs différents retours. Un énorme merci à Stefan Mengel qui a aussi passé un temps très grand à relire ce dernier document en anglais et à me faire des retours.

Tous les commentaires (mêmes typographiques, orthographiques, etc.) sur ce document sont les bienvenus et à adresser à bournez@lix.polytechnique.fr.

Sur les exercices Certains des exercices sont corrigés. Les corrections se trouvent en fin du polycopié dans un chapitre consacré à des solutions. Les exercices marqués d'une étoile nécessitent plus de réflexion.

1 Concepts mathématiques

1.1 Ensembles, Fonctions

Soit E un ensemble, et e un élément. On note $e \in E$ pour signifier que e est un élément de l'ensemble E. Si A et B sont deux ensembles, on note $A \subset B$ pour signifier que tout élément de A est un élément de B. On dit dans ce cas que A est une partie de B. Lorsque E est un ensemble, les parties de E constituent un ensemble que l'on note $\mathscr{P}(E)$. On notera $A \cup B$, $A \cap B$ pour respectivement l'union et l'intersection des ensembles A et B. Lorsque A est une partie de E, on notera A^c pour le complémentaire de A dans E.

Exercice 1 *Soient A, B deux parties de E. Prouver les lois de Morgan : $(A \cup B)^c = A^c \cap B^c$ et $(A \cap B)^c = A^c \cup B^c$.*

Exercice 2 *Soient A, B, C trois parties de E. Prouver que $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ et $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.*

Exercice 3 *(corrigé page 233) Soient A, B, C trois parties de E. Montrer que $A \cap B^c = A \cap C^c$ si et seulement si $A \cap B = A \cap C$.*

On appelle produit cartésien, de deux ensembles E et F, noté $E \times F$, l'ensemble des couples formés d'un élément de E et d'un élément de F :

$$E \times F = \{(x, y) | x \in E \text{ et } y \in F\}.$$

Pour $n \geq 1$ un entier, on note $E^n = E \times \cdots \times E$ le produit cartésien de E par lui même n fois. E^n peut aussi se définir [1] récursivement par $E^1 = E$, et $E^{n+1} = E \times E^n$.

Intuitivement, une *application* f d'un ensemble E vers un ensemble F est un objet qui associe à chaque élément e d'un ensemble E un unique élément $f(e)$ de F. Formellement, une fonction f (on parle aussi de *fonction partielle*) d'un ensemble E vers un ensemble F est une partie Γ de $E \times F$, tel que pour tout $x \in E$ il y a au plus un $y \in F$ avec $(x, y) \in \Gamma$. Son *domaine* est l'ensemble des $x \in E$ tel que $(x, y) \in \Gamma$ pour un certain $y \in F$. Son *image* est l'ensemble des $y \in F$ tel que $(x, y) \in \Gamma$ pour un certain $x \in E$. Une *application* f (on parle aussi de *fonction totale*) d'un ensemble E vers un ensemble F est une fonction dont le domaine est E.

Une *famille* $(x_i)_{i \in I}$ *d'éléments d'un ensemble* X est une application d'un ensemble I dans X. I est appelé l'ensemble des indices, et l'image par cette application de l'élément $i \in I$ est notée x_i.

Le produit cartésien se généralise à une famille d'ensembles :

$$E_1 \times \cdots \times E_n = \{(x_1, \ldots, x_n) | x_1 \in E_1, \cdots, x_n \in E_n\}.$$

1. Il y a une bijection entre les objets définis par les deux définitions.

L'union et l'intersection se généralisent à une famille quelconque de parties d'un ensemble E. Soit $(A_i)_{i\in I}$ une famille de parties de E.

$$\bigcup_{i\in I} A_i = \{e \in E | \exists i \in I \; e \in A_i\};$$
$$\bigcap_{i\in I} A_i = \{e \in E | \forall i \in I \; e \in A_i\}.$$

Exercice 4 *Soit A une partie de E, et $(B_i)_{i\in I}$ une famille de parties de E. Prouver les deux égalités suivantes*

$$A \bigcup \left(\bigcap_{i\in I} B_i \right) = \bigcap_{i\in I} (A \cup B_i)$$

$$A \bigcap \left(\bigcup_{i\in I} B_i \right) = \bigcup_{i\in I} (A \cap B_i)$$

On notera $\mathbb{N}$ l'ensemble des entiers naturels, $\mathbb{Z}$ l'ensemble des entiers relatifs, $\mathbb{R}$ l'ensemble des réels, et $\mathbb{C}$ l'ensemble des complexes. $\mathbb{Z}$ est un anneau. $\mathbb{R}$ et $\mathbb{C}$ sont des corps. On notera $\mathbb{R}^{>0}$ l'ensemble des réels positifs.

1.2 Alphabets, Mots, Langages

Nous rappelons maintenant quelques définitions élémentaires sur les mots et les langages. La terminologie, empruntée à la linguistique, rappelle que les premiers travaux sur les concepts de langages formels sont issus de la modélisation de la langue naturelle.

On fixe un ensemble fini Σ que l'on appelle *alphabet*. Les éléments de Σ sont appelés des *lettres* ou des symboles.

Exemple 1
- $\Sigma_{bin} = \{0,1\}$ *est l'*alphabet binaire.
- $\Sigma_{latin} = \{A, B, C, D, \ldots, Z, a, b, c, d, \ldots, z\}$ *est l'alphabet correspondant aux lettres de l'alphabet latin.*
- $\Sigma_{nombre} = \{0,1,2,\cdots,9\}$ *est l'alphabet correspondant aux chiffres en base 10.*
- *L'ensemble des caractères ASCII, ou l'ensemble des caractères imprimables, est un alphabet, que l'on peut noter Σ_{ASCII} (on supposera qu'il contient les symboles accentués).*
- $\Sigma_{exp} = \{0,1,2,\cdots,9,+,-,*,/,(,)\}$, *où $+$, $-$, $*$, $/$, $(,)$ désignent des symboles particuliers, est l'alphabet des expressions arithmétiques.*

Un *mot* w sur l'alphabet Σ est une suite finie $w_1 w_2 \cdots w_n$ de lettres de Σ. L'entier n est appelé la *longueur* du mot w. Il est noté length w.

Exemple 2 — 10011 *est un mot sur l'alphabet* Σ_{bin} *de longueur* 5.
- — 9120 *est un mot sur l'alphabet* Σ_{nombre}*, mais n'est pas un mot sur l'alphabet* Σ_{bin}.
- — $Bonjour$ *est un mot de longueur* 7 *sur l'alphabet* Σ_{latin}*;* $azrddfb$ *est aussi un mot de longueur* 7 *sur ce même alphabet. Elève n'est pas un mot sur cet alphabet, car le symbole è (e accent grave) n'est pas dans l'alphabet* Σ_{latin} *défini plus haut.*
- — *Éléphant et* ££z'!!!" *sont des mots sur l'alphabet* Σ_{ASCII}.
- — *243+(5*(1+6)) est un mot sur l'alphabet* Σ_{exp}.
- — *24*((((5/+)//+ est aussi mot sur l'alphabet* Σ_{exp}.

Un *langage sur* Σ est un ensemble de mots sur Σ. L'ensemble de tous les mots sur l'alphabet Σ est noté Σ^*. Le mot vide ϵ est le seul mot de longueur 0. Le mot vide n'est rien d'autre qu'un mot particulier : il est possible qu'un langage contienne le mot vide (qui est un mot particulier), ou qu'un langage ne contienne pas le mot vide. Σ^* contient toujours par définition le mot vide.

Exemple 3 — $\{0,1\}^*$ *désigne l'ensemble des mots sur l'alphabet* $\Sigma_{bin} = \{0,1\}$. *Par exemple,* $00001101 \in \{0,1\}^*$. *On a aussi* $\epsilon \in \{0,1\}^*$.
- — $\{bonjour, aurevoir\}$ *est un langage sur* Σ_{latin}. *Ce langage contient deux mots.*
- — *L'ensemble des mots du dictionnaire du français écrits sans caractères accentués est un langage sur l'alphabet* Σ_{latin}.
- — *L'ensemble des mots du dictionnaire du français est un langage sur l'alphabet* Σ_{ASCII}.
- — *L'ensemble des phrases de ce document est un langage sur l'alphabet des caractères ASCII. On observera que le caractère , c'est-à-dire le caractère espace blanc, utilisé pour séparer les mots dans une phrase, est un caractère ASCII particulier.*
- — ${\Sigma_{exp}}^*$ *contient des mots comme 24*((((5/+)//+ qui ne correspondent à aucune expression arithmétique valide. L'ensemble des mots correspondant à une expression arithmétique valide, comme 5+(2*(1-3)*3), est un langage particulier sur* Σ_{exp}.

On définit une opération de *concaténation* sur les mots : la concaténation du mot $u = u_1 u_2 \cdots u_n$ et du mot $v = v_1 v_2 \cdots v_m$ est le mot noté $u.v$ défini par

$$u_1 u_2 \cdots u_n v_1 v_2 \cdots v_m,$$

c'est-à-dire le mot dont les lettres sont obtenues en juxtaposant les lettres de v à la fin de celles de u. L'opération de concaténation notée . est associative, mais non-commutative. Le mot vide est un élément neutre à droite et à gauche de cette opération. On appelle aussi Σ^* le *monoïde* (libre) sur l'alphabet Σ (car l'opération de concaténation lui donne une structure de monoïde)

On note aussi uv pour la concaténation $u.v$. En fait, tout mot $w_1 w_2 \cdots w_n$ peut se voir comme $w_1.w_2 \cdots .w_n$, où w_i représente le mot de longueur 1 réduit à la lettre

w_i. Cette confusion entre les lettres et les mots de longueur 1 est souvent très pratique.

Exemple 4 *Si Σ est l'ensemble $\{a,b\}$, alors $aaab$ est le mot de longueur 4 dont les trois premières lettres sont a, et la dernière est b. C'est aussi la concaténation des quatre mots de longueur un : a, a, a et b.*

Lorsque i est un entier, et w un mot, on écrit w^i pour le mot obtenu en concaténant i fois le mot w : w^0 est le mot vide ϵ, w^1 est le mot w, et w^{i+1} est $w.w.\cdots,w$ où il y a i fois le mot w. Autrement dit, $w^{i+1} = w^i w = w w^i$ pour tout entier i.

Exemple 5 *En utilisant la confusion précédente entre lettres et mots de longueur 1, le mot $aaabbc$ peut aussi s'écrire a^3b^2c.*

Un mot u est un *préfixe* d'un mot w, s'il existe un mot z tel que $w = u.z$. C'est un *préfixe propre* si $u \neq w$. Un mot u est un *suffixe* d'un mot w s'il existe un mot z tel que $w = z.u$.

1.3 Changement d'alphabet

Il est souvent utile de pouvoir réécrire un mot sur un alphabet en un mot sur un autre alphabet. Par exemple, on a souvent besoin en informatique de coder en binaire, c'est-à-dire avec l'alphabet $\Sigma = \{0,1\}$.

Une façon de faire pour changer d'alphabet est de procèder par réécriture lettre par lettre.

Exemple 6 *Si Σ est l'alphabet $\{a,b,c\}$, et $\Gamma = \{0,1\}$, alors on peut coder les mots de Σ^* sur Γ^* par la fonction h telle que $h(a) = 01$, $h(b) = 10$, $h(c) = 11$. Le mot $abab$ se code alors par $h(abab) = 01100110$, c'est-à-dire par le mot obtenu en codant lettre par lettre.*

Très formellement, étant donnés deux alphabets Σ et Γ, un *homomorphisme* est une application de Σ^* dans Γ^* telle que

— $h(\epsilon) = \epsilon$

— $h(u.v) = h(u).h(v)$ pour tous mots u et v.

En fait, tout homomorphisme est parfaitement déterminé par son image sur les lettres de Σ. Il s'étend alors aux mots de Σ^* par

$$h(w_1 w_2 \cdots w_n) = h(w_1).h(w_2).\ldots.h(w_n)$$

pour tout mot $w = w_1 w_2 \cdots w_n$: les plus assidus auront remarqué par la suite que c'est en fait une conséquence du théorème 2.5 du chapitre 2.

1.4 Graphes

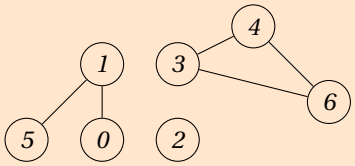
Un *graphe* $G = (V,E)$ est donné par un ensemble V, dont les éléments sont appelés *sommets*, et d'une partie de $E \subset V \times V$, dont les éléments sont appelés des *arcs*. Dans certains ouvrages, on parle de *nœuds* comme synonyme de sommet.

Un *chemin* de s à t est une suite $(s = s_0, \ldots, s_n = t)$ de sommets tels que, pour $1 \leq i \leq n$, (s_{i-1}, s_i) soit un arc. Un *chemin simple* est un chemin qui ne passe pas deux fois par le même sommet. Un *circuit* est un chemin de longueur non nulle dont l'origine coïncide avec l'extrémité.

Si les arcs ne sont pas orientés, c'est-à-dire, si l'on considère qu'à chaque fois qu'il y a l'arc (u, v), alors il y a aussi l'arc (v, u) (et réciproquement), on dit que le graphe G est *non orienté*, et les éléments de E sont appelés des *arêtes*. Lorsqu'il y a une arête entre u et v, c'est-à-dire lorsque $(u, v) \in E$, on dit aussi que u et v sont voisins. Le degré d'un sommet u est le nombre de ses voisins.

Exemple 7 *Le graphe (non-orienté)* $G = (V, E)$ *avec*
- $V = \{0, 1, \ldots, 6\}$
- $E = \{(0,1), (3,4), (5,1), (6,3), (6,4)\}$.

est représenté ci-dessous.



Un graphe est dit *connexe* si deux quelconques de ses sommets sont reliés par un chemin.

Exemple 8 *Le graphe de l'exemple 7 n'est pas connexe, car il n'y a aucun chemin entre les sommets* 1 *et* 6.

1.5 Arbres

Les arbres sont omniprésents en informatique. En fait, plusieurs notions distinctes se cachent sous cette terminologie : arbres libres, arbres enracinés, arbres ordonnés, etc.

Il y a par ailleurs plusieurs façons de présenter les arbres, et ces différentes notions. Essentiellement, on peut le faire en partant de la théorie des graphes, ou alors en partant de définitions inductives (récursives).

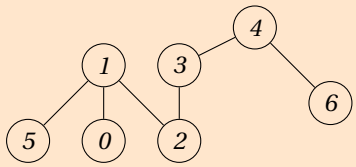
Puisque nous reviendrons sur les définitions inductives de plusieurs de ces classes d'arbres dans le chapitre 2, présentons ici les arbres en utilisant les graphes.

On va présenter dans ce qui suit des familles d'arbres de plus en plus contraints : dans l'ordre, on va présenter les arbres libres, puis les arbres enracinés, les arbres ordonnés.

Arbres libres

Un *arbre libre* est un graphe non-orienté connexe et sans circuit. On appelle *feuille* un sommet de l'arbre qui ne possède qu'un seul voisin. Un sommet qui n'est pas une feuille est appelé *un sommet interne*.

Exemple 9 (Un arbre libre) *Représentation graphique d'un arbre libre, dont les feuilles sont les sommets* $5, 0$ *et* 6.

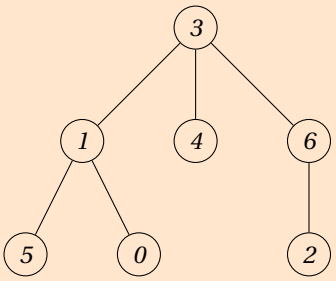


Presque tous nos arbres seront étiquetés : soit A un ensemble dont les éléments sont appelés des *étiquettes*. Un *arbre étiqueté par* A est la donnée d'un arbre $G = (V, E)$ et d'une application qui associe à chaque sommet de V un élément de A.

Arbre enraciné

Un *arbre enraciné* ou *arbre* est un arbre libre muni d'un sommet distingué, appelé sa *racine*. Soit T un arbre de racine r.

Exemple 10 (Un arbre) *Représentation graphique d'un arbre, dont les feuilles sont les sommets* $5, 0, 4$ *et* 2. *On représente la racine* 3 *en haut.*



Pour tout sommet x, il existe un chemin simple unique de la racine r à x. Tout sommet y sur ce chemin est un *ancêtre* de x, et x est un *descendant* de y. Le *sous-arbre* de racine x est l'arbre contenant tous les descendants de x. L'avant-dernier sommet y sur l'unique chemin reliant r à x est le *parent* (ou le *père* ou la *mère*) de x, et x est un *enfant* (ou un *fils* ou une *fille*) de y.

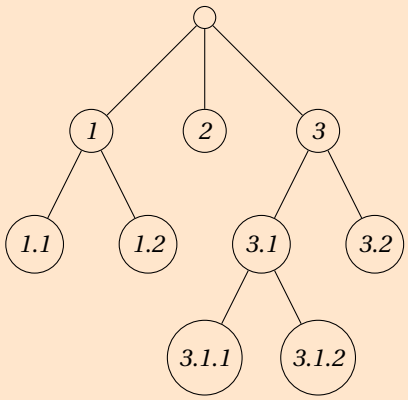
L'*arité* d'un sommet est le nombre de ses enfants. Un sommet sans enfant est une *feuille*, un sommet d'arité strictement positive est appelé *sommet interne*. La *hauteur* d'un arbre T est la longueur maximale d'un chemin reliant sa racine à une feuille. Un arbre réduit à un seul sommet est de hauteur 0.

Arbres ordonnés

Un *arbre ordonné* (on dit aussi *arbre plan*) est un arbre dans lequel l'ensemble des enfants de chaque sommet est totalement ordonné. Autrement dit, pour chaque sommet interne d'arité k, on a la notion de 1ier fils, 2ième fils, …, $k$ème fils.

Par exemple, un livre structuré en chapitres, sections, etc se présente comme un arbre ordonné.

Exemple 11 (Arbre ordonné de la table des matières d'un livre)



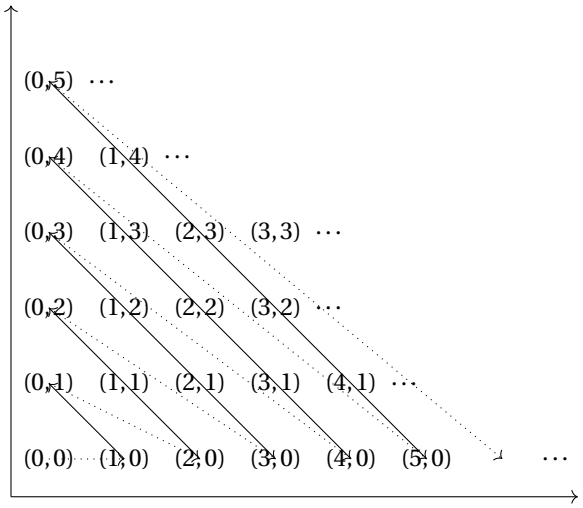
Autres notions d'arbre

Le concept *d'arbre binaire* est assez différent des définitions d'arbre libre, arbre enraciné et arbre ordonné. Il est présenté dans la section 2.4.2 du chapitre 2.

Les *termes* sont des arbres ordonnés étiquetés particuliers. Ils sont présentés dans la section 2.4.4 du chapitre 2.

2 La méthode de diagonalisation

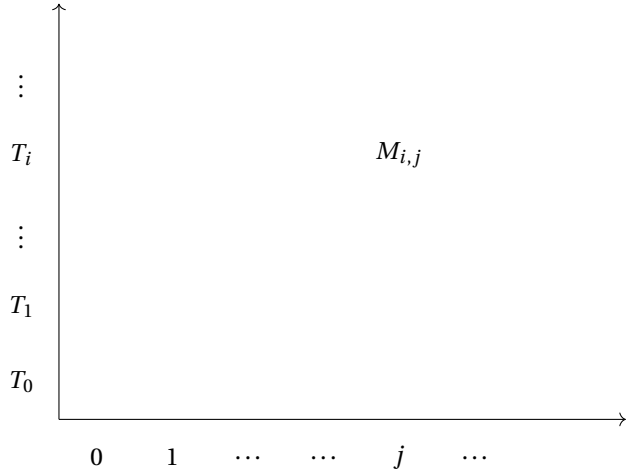
Rappelons que $\mathbb{N}^2 = \mathbb{N} \times \mathbb{N}$ est *dénombrable* : il est possible de mettre en correspondance $\mathbb{N}$ avec $\mathbb{N}^2$. Nous allons illustrer graphiquement une façon de parcourir les couples d'entiers.



Exercice 5 *(corrigé page 233) Prouver formellement que* $\mathbb{N}\times\mathbb{N}$ *est dénombrable en exhibant la bijection* $f:\mathbb{N}^2\to\mathbb{N}$ *de la figure plus haut.*
Produire une autre bijection entre $\mathbb{N}^2$ *et* $\mathbb{N}$*.*

Par contre, l'ensemble des parties de $\mathbb{N}$ n'est pas dénombrable : cela peut se prouver par la *méthode de diagonalisation* dûe à Cantor.

Illustrons-le graphiquement. Supposons que l'on puisse énumérer l'ensemble des parties de $\mathbb{N}$, et notons les T_1, T_2, $\ldots T_n$ $\ldots$Chaque partie T_i de $\mathbb{N}$ peut se voir comme la ligne i du tableau $M=(M_{i,j})_{i,j}$ à entrées dans $\{0,1\}$ dont l'élément $M_{i,j}$ est 1 si et seulement si l'élément j est dans la $i$ème partie de $\mathbb{N}$.



On considère alors la partie T^* obtenue en "inversant la diagonale de M" : formellement, on considère $T^* = \{j | M_{j,j} = 0\}$. Cette partie de $\mathbb{N}$ n'est pas dans l'énumération, car sinon elle devrait avoir un numéro j_0 : si $j_0 \in T_{j_0} = T^*$, alors on devrait avoir $M_{j_0,j_0} = 1$ par définition de M, et $M_{j_0,j_0} = 0$ par définition de T^* : ce qui est impossible. Si $j_0 \notin T^*$, alors on devrait avoir $M_{j_0,j_0} = 0$ par définition de M, et $M_{j_0,j_0} = 1$ par définition de T^* : ce qui est impossible.

Cet argument est à la base de certains raisonnements en calculabilité, comme nous le verrons.

Exercice 6 *Prouver que l'ensemble des suites $(u_n)_{n\in\mathbb{N}}$ à valeurs dans $\{0,1\}$ n'est pas dénombrable.*

Exercice 7 *Prouver que $\mathbb{R}$ n'est pas dénombrable.*

3 Notes bibliographiques

Lectures conseillées Pour aller plus loin sur les notions évoquées dans ce chapitre, nous suggérons la lecture de [Arnold & Guessarian, 2005] ou du polycopié du cours INF411 (qui portait le nom de INF421 avant 2014-2015), ou des polycopiés des cours de l'École Polytechnique de première année et des cours de classes préparatoires.

Bibliographie La partie sur les arbres est essentiellement reprise du polycopié de INF421 (version avant 2014-2015). Le reste du chapitre est inspiré de différentes sources dont le polycopié de INF561 (cours qui a été renuméroté. Je parle là du contenu du cours INF561 lorsqu'il était donné par O. Bournez), [Hopcroft & Ullman, 2000] et [Arnold & Guessarian, 2005]. L'introduction est essentiellement reprise du livre [Sipser, 1997].

Index

Bibliographie

[Arnold & Guessarian, 2005] Arnold, A. & Guessarian, I. (2005). *Mathématiques pour l'informatique*. Ediscience International.

[Hopcroft & Ullman, 2000] Hopcroft, J. E. & Ullman, J. D. (2000). *Introduction to Automata Theory, Languages and Computation, Second Edition* (2nd ed.). Addison-Wesley.

[Sipser, 1997] Sipser, M. (1997). *Introduction to the Theory of Computation*. PWS Publishing Company.