Machine, Algorithme et Programme : une introduction à la logique et à la complexité des algorithmes.

Philippe Langevin
langevin@univ-tln.fr

Author address:

LABORATOIRE GRIM, UNIVERSITÉ DE TOULON ET DU VAR

Introduction

Le présent document rassemble des notes du cours « I2 » de la Faculté des Sciences et Techniques de l'université de Toulon que j'ai dispensé au milieu des années quatre vingt dix. Il s'agit[1] d'un cours d'algorithmique et de programmation qui s'adresse aux étudiants de deuxième année du DEUG MIAS, c'est-à-dire, un public ayant une expérience de la programmation impérative : `pascal`, ou `langage C`. L'exposé privilégie le point de vue « analyse » des algorithmes (logique, complexité) au détriment des structures de données « avancées ». Les listes, graphes et arbres ne sont pas franchement abordés, les apprentis informaticiens devront suivre un cours complémentaire afin de prendre la mesure de l'équation de Nicolas WIRTH :

`algorithme` + `structure de données` = `programme.`

Au niveau le plus bas, un programme est un ensemble de boucles imbriquées agissant sur un tableau, en prendre conscience c'est réaliser le rôle des compilateurs et des langages de programmations.

Dans ce contexte dépouillé d'arbre et autres structures avancées, les problèmes de temps de calcul et certaines questions liées aux constructions récursives peuvent être abordés plus facilement, avec plus de précision et beaucoup moins de soucis d'implantation. Le passage de l'algorithme au programme que certains appellent la mise en oeuvre et d'autres l'implémentation est une des phases les plus ludiques de la pratique informatique. Quelques exercices pointus, conseils et questions sont distillés par-ci par-là pour envisager « l'art de la programmation » de Donald KNUTH. Les aspects historiques sont mis en avant pour rappeler que l'algorithmique est surtout une discipline scientifique qui s'enracine dans la Grèce antique mais se révèle totalement au XX-ième siècle.

Les cours sur les algorithmes ne manquent pas, de niveaux et de qualités très variables. L'époque des textes médiocres sur les langages et les programmes est révolue. Les titres du genre "100 programmes en `bidule`" ont finalement laissé la place à des ouvrages de bonnes qualités qui sont des échos de quelques grands classiques. Il suffit de parcourir la bibliographie pour avoir une idée de ceux qui ont ont influencé mon texte. Dans cette situation, il est particulièrement difficile de produire un document original, pour tenter de l'être un tant soit peu, j'ai décidé d'introduire un nombre limité de problématiques, choisies pour illustrer l'algorithmie en profondeur plutôt qu'en largeur. Au minimum, il me semble raisonnable d'exiger des étudiants concernés la maîtrise des solutions algorithmiques, de leurs mise en oeuvre et des temps de d'exécutions correspondants. Les développements historiques, métaphysiques sont rapportés pour motiver les troupes

[1] s'agissait puisque de réforme en réforme, le cours à déjà changé trois fois de nom, de langage d'application et de public !

vers l'analyse de deux algorithmes récursifs fondamentaux : le tri rapide et du calcul du PGCD qui constituent le point culminant de l'exposé.

1) Tracé d'un segment de droite. L'algorithme de Bresenham illustre parfaitement la finalité algorithmique : efficacité, ainsi que l'ingéniosité dont il faut souvent faire preuve pour y arriver. Un sujet qui donne lieu à des ramifications intéressantes dans le domaine de la théorie des langages.

2) Exponentiation. Une opération fondamentale, au coeur de quelques méthodes cryptographiques. La méthode d'exponentiation rapide démontre une fois de plus l'importance du principe dichotomique.

3) Évaluation d'un polynôme. La diversité des solutions algorithmiques à un problème donné est parfaitement illustrée par cette problématique simplissime. L'approche naïve donne lieu à un algorithme de complexité quadratique, $n\log(n)$ en utilisant l'exponentiation rapide mais la méthode de Hörner (linéaire) est plus performante.

4) Calcul matriciel. La résolution des systèmes d'équations linéaires est primordiale. Un calcul « naïf » de déterminant donne lieu à un temps de calcul plus qu'exponentiel. Le calcul d'un déterminant par la méthode de Gauss donne lieu à un algorithme de complexité cubique. Ses implantations, nous alertent sur les problèmes de stabilités numériques, précisions et les dépassements de capacité.

5) Ordre et tris. La plupart des algorithmes performants passent par une phase de mise en ordre des données. L'analyse du tri rapide récursif est particulièrement instructive : correction des boucles imbriquées, recherche d'instances favorables, défavorables, et calcul du temps moyen.

6) Calcul de PGCD . L'algorithme d'Euclide demeure un bon exemple pour discuter des tous les aspects : logique, temps de calculs, récursivité et suppression de la récursivité.

7) La recherche de motifs pour montrer que l'algorithmique n'est pas qu'une affaire de tables et de nombres.

Certains prétendent qu'il est possible d'apprendre l'algorithmique et même la programmation sans rien connaître au fonctionnement des ordinateurs, et du coup, ils évacuent les cours d'assembleurs. Ils ont probablement raison puisque la notion d'algorithme est bien antérieure à celle d'ordinateur. Cependant, nous ne tenterons pas de réaliser ce tour de force. Bien au contraire, utiliser le fonctionnement d'une machine est un bon moyen de jeter les bases du langage algorithmique qui rend palpable les notions de syntaxe et sémantique des instructions et, qui plus est, donne de véritables sujets de travaux pratiques : émulation d'une machine, compilation et retour à la syntaxe des instructions.

Écrire un programme pour résoudre un problème c'est bien mais encore faut il savoir prouver la correction du code, et surtout prévoir son temps d'exécution. La logique de Hoare donnera des bons réflexes aux futurs mangeurs de pizzas : les programmeurs. Les mécanismes de preuves et les temps de calcul seront vus en travaux-pratiques, au travers des quelques morceaux choisis comme : la recherche de motifs, la compression de données, construction de nombres premiers etc. . .

Finalement, il en résulte un document à l'usage des futurs informaticiens, des futurs mathématiciens et donc à tout ceux qui opteront pour une (non)-orientation math-info.

Ph. Langevin, été 2000.

Aprés cinq ans de travail, pas trés acharné mais quand même, le document est toujours en chantier. Un petit peu comme dans la philosophie linuxienne décrite dans « la cathédrale et le bazar » de Eric S. Raymond (facile à trouver sur le web), il est grand temps de le mettre à disposition des utilisateurs. Le document est libre, vous pouvez en faire ce qu'il vous plait, pour ma part, je suis à l'écoute de vos commentaires, corrections et suggestions,

Ph. Langevin, Automne 2001.