HAL
archives-ouvertes.fr

L'informatique dans les classes préparatoires aux grandes écoles

Bruno Petazzoni

▶ **To cite this version:**

Bruno Petazzoni. L'informatique dans les classes préparatoires aux grandes écoles. Revue de l'EPI (Enseignement Public et Informatique), EPI, 2001, pp.185-193. edutice-00001300

HAL Id: edutice-00001300
https://edutice.archives-ouvertes.fr/edutice-00001300

Submitted on 18 Nov 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

L'INFORMATIQUE DANS LES CLASSES PRÉPARATOIRES AUX GRANDES ÉCOLES

Bruno PETAZZONI

De 1987 à 1995

En 1987, un enseignement d'informatique est introduit dans les classes préparatoires aux Grandes Écoles d'ingénieurs (« maths sup » et « maths spé »). Il s'agit d'une initiation à l'algorithmique et à la programmation, dispensée aux étudiants sous la forme de séances de travaux pratiques, à raison d'une heure hebdomadaire.

Le langage choisi est **Pascal**, ou plus précisément **Turbo-Pascal**. Un plan d'équipement permet d'équiper rapidement les lycées concernés, avec des machines typiques de cette époque : unité centrale à base de 286, disque dur de 20 Mo, écrans monochromes, imprimantes matricielles 9 aiguilles.

Un logiciel intéressant suivra bientôt : il s'agit d'une bibliothèque, co-écrite par des universitaires et des professeurs de C.P.G.E. et facilitant la manipulation des vecteurs, des matrices et surtout des expressions. Il est possible d'introduire une expression mathématique au clavier, pour en demander l'évaluation ou, par exemple, l'intégration sur un intervalle donné.

À partir de 1989, la discipline informatique sera sanctionnée aux concours, mais très inégalement. L'E.N.S. de Lyon proposera chaque année une épreuve écrite, dont la difficulté ira croissant, le summum étant atteint en 1994 (pavages d'échiquiers) et en 1995 (analyse de l'algorithme *union-find* avec pondération et compression de chemins).

L'E.N.S. de Paris préférera proposer une épreuve orale, très innovante par sa forme : les candidats disposent de trois heures (et d'un ordinateur) pour étudier un sujet, l'analyser et le programmer ; ils rendent un dossier écrit et une disquette. On trouvera des exemples des

questions posées dans les deux tomes de *Mathématiques et informatique*, publiés chez McGraw-Hill.

L'École polytechnique adoptera une forme analogue, avec préparation « en loge » et remise d'un dossier. En revanche, les grands concours communs ne suivront pas le mouvement : parfois, au détour d'une épreuve de mathématiques appliquées, on a vu surgir une question demandant la rédaction d'un algorithme ; mais les épreuves d'informatiques à proprement parler se comptent sur les doigts de la main de Bart Simpson.

De 1995 à nos jours

À la rentrée de septembre 1995, une réforme profonde est mise en œuvre dans les C.P.G.E. : le trait le plus marquant est la mise en place de filières dès la première année. Les dénominations « maths sup » et « maths spé » sont abandonnées. L'enseignement d'informatique est entièrement revu. Il comportera désormais deux aspects :

- un enseignement de tronc commun, visant à initier les étudiants à l'algorithmique, à la programmation, et à l'utilisation d'un logiciel de calcul formel ; cet enseignement sera dispensé à raison d'une heure hebdomadaire de travaux dirigés pendant le premier trimestre de la première année, et accompagné d'une heure hebdomadaire de travaux pratiques (devant des ordinateurs, donc) pendant les deux années. Certaines de ces heures peuvent être consacrées à l'utilisation de logiciels « disciplinaires », en physique, chimie ou sciences industrielles ;
- un enseignement optionnel, proposé uniquement aux étudiants de la filière « MP » (mathématiques et physique) : dispensé à raison d'une heure de cours et d'une heure de travaux dirigés chaque semaine, après le premier trimestre de la première année, cet enseignement est complété par des heures de travaux pratiques (prises sur le contingent évoqué au paragraphe précédent).

L'enseignement de tronc commun

Cet enseignement s'appuie sur l'utilisation d'un logiciel de calcul formel : **Maple** ou **Mathematica**, au choix des équipes pédagogiques. **Maple** a été choisi majoritairement, sans que des raisons rationnelles puissent étayer ce choix. On notera qu'aucune procédure n'a été mise en place pour acheter ces logiciels à un prix raisonnable.

Chacun de ces logiciels dispose d'un langage de programmation ; mais ce langage n'a pas du tout été conçu pour une initiation à l'algorithmique et à la programmation. Il sera difficile de donner de bonnes habitudes aux étudiants avec ces outils.

L'évaluation de cet enseignement aux concours n'a encore jamais eu lieu. Une évolution se dessine toutefois : l'École polytechnique, inquiète de l'effondrement des compétences en programmation de ses étudiants depuis 1996, a décidé d'imposer à tous les candidats n'ayant pas suivi l'enseignement optionnel (décrit en détail plus loin) une épreuve d'algorithmique et de programmation, à partir de la session de 2002. Cette décision a provoqué d'assez vifs remous, aussi bien dans les associations d'enseignants que dans les autres écoles. Il est vraisemblable toutefois que ces dernières suivront le mouvement : le passé nous enseigne que, régulièrement, ce que ces écoles n'avaient pas l'envie ou les moyens d'enseigner se retrouvait au programme de C.P.G.E. : on comprendra assez facilement qu'en l'an 2000, un professeur d'informatique dans une Grande École considère l'enseignement de l'art délicat du placement des points-virgules comme indigne de son statut.

L'enseignement optionnel

Cet enseignement n'est ouvert que dans une cinquantaine de lycées. Le programme publié est précis : en première année, on étudie les méthodes de programmation (itération et récursion, diviser pour régner), les structures de données linéaires (piles et listes), et enfin un tryptique comportant logique, fonctions booléennes et circuits combinatoires. En deuxième année, on étudie un modèle de calcul simple (les automates finis), une structure de données non linéaires (les arbres binaires, et en particulier les arbres binaires de recherche) et des notions élémentaires de complexité des algorithmes.

Ces notions correspondent, peu ou prou, à celles enseignées dans tout cursus de niveau équivalent à la surface du globe. Il suffit au lecteur de visiter les sites Web des universités américaines pour s'en convaincre. Un seul absent de marque : la théorie des graphes.

Mais le programme exposé précédemment « tient » juste dans la cinquantaine d'heures de cours ; il n'était pas envisageable de le développer davantage.

La mise en oeuvre sera faite, au choix de l'équipe pédagogique, en langage **Pascal** ou **Caml** ; ce dernier est un langage de programmation

fonctionnelle (comme **Lisp**) avec un mécanisme d'inférence de type offrant une grande sécurité : il est impossible de donner la valeur 0 à un booléen, pour prendre un exemple ! Ce langage, très proche des mathématiques, est d'une grande concision et, surtout, permet de prouver facilement les programmes. Signalons qu'il est possible de programmer sans « variables » et sans boucles : ceci surprendra peut-être les habitués de la programmation impérative. Ils trouveront en annexe un exemple de programme et pourront écrire l'analogue dans leur langage préféré, aux fins de comparaison.

La première année, les deux langages ont été choisis à peu près également ; au cours des années suivantes, **Pascal** a progressivement perdu des adeptes.

À partir de 1997, les divers concours ont proposé une épreuve d'informatique aux étudiants qui avaient suivi cet enseignement. L'E.N.S. de Lyon a, courageusement, prélevé 12 places sur le concours « maths » pour ouvrir un concours « informatique » ; les deux autres E.N.S. n'ont pas encore suivi dans cette direction, mais la quasi-totalité des candidats au concours « mathématiques » de ces écoles ont suivi l'enseignement optionnel d'informatique.

Curieusement, l'E.N.S.I.M.A. de Grenoble, qui forme surtout des informaticiens, continue à recruter sur le Concours commun polytechnique (regroupement des E.N.S.I.) sans proposer à ses candidats une épreuve spécifique d'informatique.

La durée de ces épreuves varie entre deux et quatre heures. Les concours des écoles d'ingénieurs posent systématiquement des questions de programmation ; les épreuves sont rédigées de façon à mettre les candidats sur un pied d'égalité, quel que soit celui des deux langages qu'ils ont choisi. La puissance d'expression de **Caml** donne cependant un net avantage à ceux qui l'ont appris et compris.

L'avenir

L'enseignement optionnel d'informatique est confronté à de multiples problèmes. Le premier est le manque de personnel pour assurer cet enseignement : qu'un collègue parte en retraite, ou tombe malade, et son remplacement est problématique. L'absence de concours de recrutement dans cette discipline explique en partie cette situation, et la disparition de l'épreuve écrite à options, à l'agrégation de mathématiques, a supprimé un des rares moyens qui permettaient de faire découvrir l'informatique aux futurs enseignants de mathématiques.

On peut aussi trouver des motifs dans l'attraction très vive exercée sur les jeunes diplômés par le secteur informatique ; et ce ne sont pas les affirmations péremptoires d'un précédent ministre, concernant les mathématiques et l'informatique, qui ont pu améliorer la situation, bien au contraire !

Il serait souhaitable que cet enseignement soit étendu à la filière « PSI » (physique et sciences de l'ingénieur) : la nécessité de l'outil informatique pour les ingénieurs issus de cette filière (menant entre autres aux E.N.S.A.M. et formant pour l'essentiel de futurs ingénieurs producticiens) n'est pas à prouver. L'absence de moyens humains sera le premier obstacle ; le deuxième sera la nécessité de faire place à cette discipline, en supprimant des heures d'une autre matière : bon courage à ceux qui devront arbitrer !

L'observateur ne peut quand même manquer de se poser une question : dans une période où les entreprises manquent désespérément de spécialistes en informatique, est-il sage de continuer à former des bataillons de chimistes dont la moitié, à peine diplômés, sont hâtivement reconvertis, en informaticiens justement ?

Bruno PETAZZONI

ANNEXE

Un petit problème

Voici un mobile :

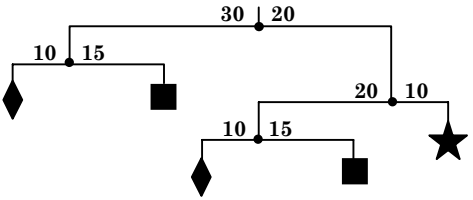

Figure 1 : un mobile.

Il est constitué de tiges horizontales, de fils verticaux, et d'éléments décoratifs : losanges, carrés, étoiles. Les masses des tiges et des fils sont négligeables ; les éléments décoratifs ont des masses connues. Les longueurs des tiges sont indiquées ; on veut *équilibrer* le mobile, c'est-à-dire déplacer chaque tige vers la gauche ou vers la droite pour obtenir l'égalité $m_g \times l_g = m_d \times l_d$, où m_g (resp. m_d) est la masse du sous-mobiles accroché à l'extrémité gauche (resp. droite) de la tige, et l_g (resp. l_d) la longueur de la partie gauche (resp. droite) de la tige.

Si les longueurs et les masses étaient des nombres réels, ceci ne présenterait pas de difficulté, grâce aux formules

$$l'_g = \frac{m_g l_g}{m_g + m_d} \text{ et } l'_d = \frac{m_d l_d}{m_g + m_d}$$

Mais, dans notre cas, longueurs et masses sont des entiers. Notant $[x]$ le plus grand entier inférieur ou égal à x, nous choisirons donc parmi les deux solutions

$$l'_g = \left[\frac{m_g l_g}{m_g + m_d} \right] \quad l'_d = l_g + l_d - l'_g$$

$$l'_d = \left[\frac{m_d l_d}{m_g + m_d} \right] \quad l'_g = l_g + l_d - l'_d$$

celle qui minimise l'erreur $\left| m_g \times l_g - m_d \times l_d \right|$. Cette quantité est proportionnelle au moment résultant au point d'accrochage de la tige.

Définitions de types

Cette analyse effectuée, passons à la programmation. Commençons par définir deux types, l'un pour les éléments décoratifs, l'autre pour les mobiles :

```
type élément_décoratif = Carré | Losange | Étoile;;

type mobile = simple of élément_décoratif |
  compliqué of int*int*mobile*mobile;;
```

Le type `élément_décoratif` est un *type union*, analogue aux types énumérés de Pascal et C ; le point de vue le plus simple est de le considérer comme un ensemble fini. Le type mobile est lui aussi un type union, analogue aux enregistrements avec parties variantes de Pascal ; il est la réunion de deux ensembles :

- le premier est, non pas le type `élément_décoratif`, mais son image par une fonction injective ; cette subtilité évite de confondre un élément décoratif (par exemple `Carré`) avec le mobile réduit à une fil auquel est suspendu cet élément (à savoir `compliqué Carré`) ;
- le deuxième ensemble est l'image par une autre fonction injective d'un produit cartésien de quatre types (donc de quatre ensembles) ; on décrit ainsi un mobile non réduit à un élément de décor, par la donnée des longueurs des deux morceaux de la tige accroché au fil « du haut », et des deux mobiles suspendus aux extrémités de la tige.

Voici maintenant la façon dont nous définirions le mobile de la figure 1 :

```
let m1 = compliqué(10,15,simple Losange,simple Carré) in
let m = compliqué(30,20,m1,compliqué(20,10,m1,simple
  Étoile));;
```

La syntaxe `let m = ...` crée une *liaison* entre le nom `m` et la valeur décrite par l'expression à droite du signe =. Notez bien que `m` n'est pas une « variable » : la liaison est définitive. Au passage, le nom `m` désigne un objet de type `mobile` : le typage est effectué automatiquement par Caml. La liaison `let m1 = ...` est locale à la liaison de `m` : de l'extérieur, on ne voit que cette dernière.

Rédaction d'une première fonction

Avant de résoudre notre problème, nous allons, pour l'exemple, rédiger une fonction qui calcule la masse totale d'un mobile, en s'appuyant sur la connaissance des masses des éléments décoratifs : celles-ci seront décrites par une fonction que voici :

```
let masse_élément_décoratif = function
  | Carré -> 2
  | Losange -> 1
  | Étoile -> 3;;
```

La syntaxe est assez facile à comprendre : chaque « barre verticale » introduit un cas de figure. Le type de cette fonction est `élément_décoratif -> int`, ce qui se comprend aisément si l'on sait que `int` est le type (prédéfini) des entiers en Caml.

Nous en déduisons une fonction qui calcule la masse totale d'un mobile ; cette fonction attend deux arguments : le premier est la fonction qui joue le rôle de « catalogue » des masses des éléments décoratifs ; le deuxième est le mobile considéré.

```
let rec masse_mobile p = function
 | simple x -> p x
 | compliqué(_,_,g,d) -> masse_mobile p g + masse_mobile p d;;
```

Les initiés reconnaissent un parcours en profondeur d'un arbre binaire. Pour obtenir la masse totale du mobile `m` dessiné à la figure 1, il suffit de demander :

```
masse_mobile masse_élément_décoratif m;;
```

À propos de l'appel de fonction en Caml

Une petite note explicative : les parenthèses ne sont pas requises lorsque l'on applique une fonction ; ainsi, $f(x)$ peut être noté `f x`. Pour les « fonctions de plusieurs variables », il existe deux notations différentes ; voyons ceci sur l'exemple de la fonction $s : (x, y) \mapsto x^2 + y^2$. La première syntaxe, très proche des mathématiques, existe en deux variantes :

```
let s = function (x,y) -> x*x+y*y ;;
let s (x,y) = x*x+y*y ;;
```

La deuxième syntaxe est radicalement différente :

```
let s x y = x*x+y*y ;;
```

L'interprétation est la suivante : `s x` est la fonction qui, à y, associe $x^2 + y^2$. Cette syntaxe d'apparence surprenante se révèle d'une efficacité redoutable.

Venons-en au cœur du problème!

Entrons dans le vif du sujet. Nous allons d'abord rédiger la fonction qui « redécoupe » au mieux la tige ; elle attend quatre arguments l_g, l_d, m_g et m_d et

rend le couple formé par les longueurs ajustées. Le lecteur attentif notera l'absence de valeurs absolues ; le lecteur perspicace expliquera cette absence !

```
let ajuste lg ld mg md =
  let l=lg+ld and m=mg+md in
  let lgl = md*l/m in
  let ldl = l-lgl and lg2 = lgl+l in
  let ld2 = ldl-l in
    if md*ldl-mg*lgl < mg*lg2-md*ld2
      then (lgl,ldl) else (lg2,ld2) ;;
```

Tout est en place pour la grande scène finale. La fonction `reequilibre` attend deux arguments : le premier est la fonction donnant la masse de chaque élément décoratif, le deuxième est le mobile. Le résultat rendu est le mobile équilibré.

Si le mobile se réduit à un élément décoratif, aucun travail n'est requis. Sinon, il faut successivement :

1. Équilibrer les sous-mobiles gauche et droit ;
2. Calculer la masse de chacun de ces sous-mobiles, pour ajuster le point d'accrochage de la tige sur le fil.

Bien entendu, il y a de la récursivité dans l'air. D'autre part, il est clair que, lors de l'équilibrage d'un sous-mobile non réduit à un élément décoratif, on calcule sa masse (sans le dire). Cette remarque permet de n'effectuer qu'un seul parcours de l'arbre sous-jacent à notre mobile. Nous rédigeons donc une fonction auxiliaire, justement baptisée `aux` [1] qui équilibre un mobile et calcule sa masse ; cette fonction rend le couple formé du mobile équilibré et de sa masse. `fst` est une fonction de bibliothèque qui rend la première composante d'un couple.

```
let rééquilibre_mobile p mon_mobile =
  let rec aux = function
  | simple x as mob -> (mob,p x)
  | compliqué (lg, ld, g, d) ->
     let (g',mg) = aux g and (d' ,md) = aux d in
     let (lg',ld') = ajuste lg ld mg md in
     (compliqué(lg',ld',g',d'),mg+md)
  in fet (aux mon_mobile) ;;
```

Pour des renseignements complémentaires sur Caml, visitez la page Web : http://caml.ioria.fr/index-fra.html

1. Note aux « stagiaires lourds » de la décennie 80-90 : nous aurions pu l'appeler `toto`.