Vous avez dit « Algorithme » ?

Jean-Claude Oriol(*)

1. Introduction

L'algorithmique a rempli une partie de mon activité scientifique et mes activités d'enseignement pendant un grand nombre d'années(1).
Il est indéniable que les activités désignées par le mot « Algorithme » existent depuis fort longtemps ; mais il y a eu sans doute un temps très long entre des algorithmes « en actes » et la prise de conscience du fait que l'on fait un algorithme.
Ainsi il est de tradition de dater les premiers algorithmes à la période Babylonienne et à la description de divers calculs de taxes (eh oui déjà…) et on en trouvera un certain nombre dans l'ouvrage de Jören Friberg (FRIGBER, 2007).
Mais sans doute, l'algorithme d'Euclide, (≈ 325 av J.-C. - ≈ 265 av J.-C. in RONAN (1988)), de recherche du pgcd est parmi ceux qui ont marqué notre formation en mathématique ; j'ose en rappeler le principe(2) :

***a* et *b* sont deux entiers positifs avec $a \geq b$,**
si $a = b$ alors pgcd(a, b) = a
sinon
si $b \geq a - b$ alors pgcd(a, b) = pgcd(b, $a - b$)
sinon pgcd(a, b) = pgcd($a - b$, b)

Par exemple : pgcd(35, 14) = pgcd(21, 14) = pgcd(14,7) = pgcd(7,7) = 7.

Bien entendu tout le monde connaît la forme qui converge plus rapidement à l'aide du reste r de la division de a par b.

Mais la forme par soustractions successives est plus proche de la forme primitive puisque le point de départ d'Euclide était un problème géométrique.
Dans le graphique ci-dessous le point C a pour coordonnées (35,14). On enlève au rectangle ACBD un carré de coté 14 et le rectangle qui reste a des côtés de longueurs 21 et 14. On enlève un deuxième carré de côté 14, et le rectangle qui reste a des cotés de longueurs égales à 7 et 14. On enlève un carré de côté 7 et nous avons enfin un carré $A_1C_1B_2D$.
La longueur du côté de ce dernier carré est égale au pgcd cherché.

(*) Université Lyon2, Centre de recherche CERRAL. IUT Lumière, Département Statistique et Informatique Décisionnelle.
(1) On trouvera quelques éléments concernant mes activités dans ce domaine à l'adresse suivante : https://sites.google.com/site/jeanclaudeoriol/Home/activits-algorithmiques-et-informatique.
(2) Dans cet article pour décrire les algorithmes je vais utiliser une espèce de pseudo langage que j'avais baptisé ailleurs « pidgin pascal » et qui devrait être accessible à tous.

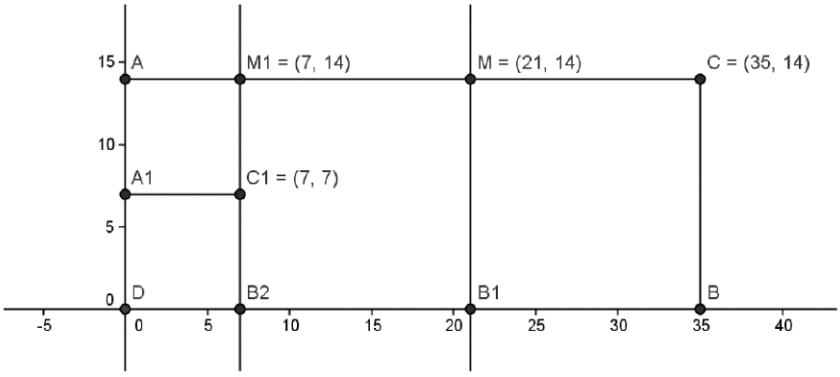

Figure 1 : pgcd de deux entiers (35 et 14)

Ce passage de l'arithmétique au champ géométrique est à n'en pas douter un « jeu de cadres » comme les a repérés, soulignés et analysés Régine Douady (in Douady R., 1986).
D'autres activités du même genre peuvent être proposées et on pense bien sûr au rectangle d'or. Et dans le même registre, l'IREM de Montpellier (IREM Montpellier, 2009) propose une activité géométrique rapprochant l'algorithme d'Euclide et les fractions continues sous le titre « *Anthyphérèse : Algorithme géométrique unifiant l'algorithme d'Euclide et l'algorithme des fractions continues* » à l'aide de Geoplan.

2. Le jeu de la polysémie

Nous venons de le voir, les mathématiques se sont penchées sur le berceau de la naissance du mot algorithme. On sait que le nom est issu du latin médiéval *algorithmus* (1554) forme latinisée du nom du mathématicien musulman perse Al-Khawarizmi (≈780, ≈ 850) (in Le Robert, 1986, page 48).
Pour ceux que la norme rassure, l'AFNOR définit un algorithme comme « l'ensemble des règles opératoires et des procédés définis en vue d'obtenir un résultat déterminé au moyen d'un nombre fini d'opérations ».
Il est peut être délicat de donner cette définition aux élèves !
Voilà comment nous nous adressions à des élèves du secondaire 4 et 5 au Québec dans un ouvrage couvrant le cours d' « *Introduction à la science de l'informatique* » à leur intention : « *Cette formulation qui est en quelque sorte une recette, une liste précise et ordonnée d'actions à faire exécuter par la machine s'appelle un algorithme* » (in Arcouet M., Oriol J.-C., 1986, page 28).

Revenons à une perspective plus générale ; ces vingt dernières années nous avons pu observer dans l'activité scientifique une inflation du mot algorithme qui passe des tours de Hanoï à divers algorithmes de tri, à celui des huit dames, ou bien l'algorithme du simplexe, etc. et nous sommes tous plus ou moins consommateurs parfois sans le savoir des algorithmes génétiques, du fameux garbage collector (allocation de mémoire et ramasse miettes), des algorithmes de cryptologie ou de compression de données.

Et parmi ceux que je préfère il y a les algorithmes gloutons ; ce sont des algorithmes qui choisissent la meilleure solution étape par étape, mais cette solution n'est pas forcément la meilleure globalement.
Un exemple qui peut être compris facilement est celui du rendu de monnaie. On rend la monnaie en choisissant toujours la pièce la plus forte (ainsi si l'on veut rendre 6 on rendra d'abord une pièce de 5) et la solution optimale cherchée est de rendre la monnaie avec le moins de pièces possible. Dans le système de pièces européen (en centimes : 1, 2, 5, 10, 20, 50, 100, 200), on peut montrer que l'algorithme glouton donne toujours une solution optimale. Mais si le système de pièces était (1, 3, 4), l'algorithme glouton ne serait pas optimal globalement, en effet pour un rendu de monnaie égal à 6 l'algorithme glouton donnera 4, 1 et 1 alors que 3, 3 est une solution optimale[3].

Mais l'usage du mot « algorithme » dépasse ainsi largement le champ des mathématiques ainsi qu'en atteste le dictionnaire Le Robert qui finit son article par « enchaînement des actions nécessaires à l'accomplissement d'une tâche ».
Pour ne citer que le fameux « algorithme de la pâte à crêpes », Google renvoie 1 520 occurrences en 0.26 secondes !

3. Et l'informatique là-dedans ?

On vient de le voir l'algorithme existe sans les mathématiques (il y avait peut être un algorithme de la pierre taillée, puis un autre de la pierre polie ?) et sans l'informatique.

A. Quelques suppléments apportés par l'informatique

En revanche l'informatique et l'algorithmique dialoguent et s'enrichissent mutuellement. Aurait-on par exemple introduit dans les programmes une activité algorithmique, sans l'omniprésence de l'informatique dans la société ?
Citons quelques-uns des enrichissements de l'algorithmique par l'informatique :

1. Terminaison des algorithmes (en informatique les algorithmes doivent se terminer).
2. Preuve des algorithmes : généralement obtenue par l'écriture de l'algorithme sous forme récursive, puis l'écriture d'un invariant.
3. Étude de la complexité des algorithmes[4] (sans vouloir les citer tous nous avons O(1) complexité constante, O(log(n)) complexité logarithmique, O(n) complexité linéaire, O(n log(n)) complexité quasi-linéaire, O(n^2) complexité quadratique, etc.)

(3) Les références de sites Internet sont souvent sujettes à débats et parfois donnent des explications erronées ou incomplètes mais vous pourrez trouver des algorithmes gloutons à l'adresse suivante : http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/CM2009/Gloutons/AlgorithmesGloutons6.pdf

(4) Dans ce paragraphe j'utilise la notation O dont voici une définition dans le cadre algorithmique :
« Soient f et g deux applications de l'ensemble des entiers naturels dans l'ensemble des réels.
Notation O : on note $f = O(g)$, et on prononce « f est un grand O de g » pour exprimer qu'il existe un entier naturel n_0 et un réel $a > 0$, tels que pour tout $n > n_0$ on ait $f(n) \leq a.g(n)$ »
in Lévy (1994)

4. Le point précédent est un élément important de l'activité algorithmique car on décidera d'effectuer l'algorithme si on a une chance « d'arriver au bout » dans le temps imparti et cette considération est facile à constater quand on effectue des algorithmes de tri.
5. Évaluation de l'efficacité d'un algorithme : il faut pour cela avoir un modèle concernant les données et faire appel à des méthodes appelées « combinatoire analytique ».

B. Un exemple de preuve et de terminaison

Intéressons-nous aux deux premiers points : la preuve et la terminaison.
Supposons que nous voulions construire un algorithme qui fasse l'addition de deux entiers positifs ou nuls a et b.
À notre disposition nous n'avons que la fonction Successeur (+1) et la fonction Prédécesseur (−1).
Nous écrivons

Somme(a,b) = {si b = 0 alors a sinon Somme(Successeur(a), Prédécesseur(b))}

Preuve de terminaison : b est positif ou nul. Quand il est nul, l'algorithme s'arrête immédiatement et on a le résultat. S'il n'est pas nul, c'est un entier positif et à chaque appel de la fonction, b est remplacé par le prédécesseur de b, donc par $b - 1$ et forcément au bout d'un nombre b d'appels de la fonction b sera égal à 0.

Preuve par récurrence et invariant : si le résultat est vrai pour **Somme (Successeur(a),Prédécesseur (b))** alors il est vrai pour **Somme(a,b)** car pour b non nul :
Somme(a,b) = Somme(Successeur(a),Prédécesseur (b))
Ou encore $a + b = (a + 1) + (b - 1)$.

C. Codage de l'algorithme

On peut essayer de coder un tel algorithme dans un langage quelconque. Prenons l'exemple d'une feuille Excel.
Si je veux ajouter 10 et 12 :

a=	10	b=	12
	11		11
	12		10
	13		9
	14		8
	15		7
	16		6
	17		5
	18		4
	19		3
	20		2
	21		1
	22		0

On a ainsi 12 lignes de calculs (plus la première ligne des données) ; remarquons que la somme de 12 et 10 n'aurait donné lieu qu'à 10 lignes de calculs.

Les formules dans les cellules sont très simples, en voici un aperçu :

A	B	C	D
a=	10	b=	12
	=SI(D1>0;B1+1;"")		=SI(D1>0;D1-1;"")
	=SI(D2>0;B2+1;"")		=SI(D2>0;D2-1;"")
	.....		.....

Les lignes suivantes sont obtenues en recopiant ces cellules vers le bas.

On peut ainsi travailler « sérieusement » avec des élèves, c'est-à-dire dépasser le côté « ça marche ou pas » que le codage des algorithmes prend parfois.

Tiens, à titre de devoir à faire à la maison :

1. Construire l'algorithme et coder dans un langage au choix (Excel pourquoi pas) la fonction Prédécesseur (on n'a pas droit à −1 bien sûr) à partir de la fonction Successeur (bon ici on a droit à +1).
2. Construire la fonction Successeur : on n'a pas droit à +1 mais on a droit aux fonctions sur les chaînes de caractères.
3. Construire l'algorithme du produit de deux entiers positifs (ici on a droit à la fonction Somme précédente).
4. Construire l'algorithme de la fonction *a* puissance *b* de deux entiers positifs (ici on a droit à la fonction Somme précédente).
5. Construire un algorithme de calcul en multi précision.

4. Complexité des algorithmes et un exemple de tri

A. Introduction

Les tris ont été, et sont toujours une activité importante de la science algorithmique. Une personnalité marquante de la programmation D. Knuth a écrit un ouvrage de référence « The Art of Computer Programming » dont le volume 3 est entièrement consacré aux tris (KNUTH D., 1998).
Traiter ici de la complexité des algorithmes (de tri par exemple) serait bien imprudent tant le sujet est vaste et mérite une attention soutenue.
Je donne seulement deux points d'entrées concernant la complexité et un calcul sur un exemple simple :

- Le premier point d'entrée concerne le nombre d'opérations que l'algorithme devra faire pour trier n éléments. : on donne soit le nombre maximum d'opérations soit une complexité moyenne (ce qui est toujours plus difficile à calculer).
- Le second concerne l'utilisation de la mémoire que l'on appelle alors complexité spatiale.

Pour les tris les plus simples (ceux auxquels on pense spontanément), il est assez facile de calculer le nombre de comparaisons qui est de l'ordre de n^2, et les meilleurs tris « sont en $n \times \log(n)$ ».

B. L'exemple du tri par bulles

Illustrons le genre de calcul mis en place par le « tri par bulles » de données dans un tableau :
Je rappelle, en pseudo-code, l'algorithme du tri bulle :

```
PROCEDURE TriBulle (Tableau a[1:n])
  VARIABLES Permut : Booleen ; Borne : entier
        Permut = VRAI
        Borne = n
        TANT QUE (Permut = VRAI)
              Permut = FAUX
              Borne = Borne - 1
              POUR i VARIANT DE 1 à Borne FAIRE
                    SI a[i] > a[i+1] ALORS   echanger(a[i],a[i+1])
                                             Permut = VRAI
                    FIN SI
              FIN POUR
        FIN TANT QUE
FIN PROCEDURE TriBulle
```

Pour ceux qui n'auraient jamais rencontré cet algorithme voici l'exemple de ce qu'il fait sur un tableau (de dimension modeste) :

1	2	3	4	5	Commentaire
7	3	1	8	6	données
3	7	1	8	6	Échange a(1) et a(2)
3	1	7	8	6	Échange a(2) et a(3)
3	1	7	8	6	Pas d'échange
3	1	7	6	8	Échange a(4) et a(5)

Figure 2 : début d'un tri par bulles

Le cinquième élément est trié, c'est comme une bulle qui serait « remontée » d'où le nom du tri[5] et on va recommencer avec le tableau de dimension 4.

C. Complexité algorithmique et complexité spatiale

On calcule facilement que le nombre **maximum** d'opérations est égal à :

- 3 pour chaque boucle « tant que » (Permut = FAUX et gestion de l'entier Borne c'est à dire Borne=Borne -1 et test) c'est-à-dire un maximum de $3 \times n$.
- à l'intérieur de la boucle il y a :
  - 1 test,
  - 1 échange éventuel c'est-à-dire 3 opérations (échanger A et B c'est {C ← A, A ← B, B ← C }),

(5) Notons que si nous voulions « prouver » ce programme et dégager les invariants nous l'écririons sous une forme fonctionnelle dans le style :
Trier (a[1,n]) = {si n = 1 alors a[1 ;n]
sinon (a[1,n - 1] = (a[1,n] \ Max(a[1,n]))) Trier (a[1,n - 1]) & Max(a[1,n])}

- une affectation éventuelle (Permut = VRAI),
- c'est à dire 5 opérations.
- Ces 5 opérations sont faites $(n-1)+(n-2)+\ldots+1$ fois c'est-à-dire $n\times(n-1)/2$ fois.
- Donc en tout $5\times n\times(n-1)/2$ opérations.

Donc, en comptant tout, le nombre maximal d'opérations[6] est égal à

$$3\times n+5\times n\times(n-1)/2.$$

Les constantes n'ont pas d'importance et donc la complexité algorithmique du tri est « en n^2 » ; on peut s'en convaincre sur le cas le plus « défavorable par exemple :

1	2	3	4	5	
8	7	6	3	1	données

Cette complexité donnera une idée du temps mis pour trier ce tableau ; pour s'en convaincre, prenez une colonne de 50 000 cellules contenant des nombres aléatoires et triez le dans Excel cela vous donnera une idée de l'efficacité de la fonction TRI d'Excel.

La complexité spatiale utilise :

- n cellules pour stocker les données,
- n cellules pour avoir les indices (car j'ai supposé les données dans un tableau),
- une cellule pour la variable Permut, une cellule pour la variable i et une pour la variable Borne,
- et une cellule pour faire l'échange,

donc en tout $2n+4$ cellules.

Autrement dit la « **complexité spatiale est de l'ordre de *n*** ».

D. Un autre exemple de tri : tri par arbre binaire ou tri des bijoutiers

Le nombre de tris référencés[7] est considérable ; le choix d'un tri particulier est dicté par la structure des données à trier d'où l'importance de ce champ en informatique.

Prenons un exemple du tri par arbre binaire appelé aussi tri des bijoutiers.

Considérons le problème du bijoutier voulant trier par grosseur un tas de diamants : pour faire cette opération il se sert d'un tamis ce qui lui permet de séparer le tas initial en deux, et il recommence avec de nouveaux tamis pour chaque tas. On le comprend facilement l'efficacité du tri est fonction des trames des tamis utilisés. Appliquons notre algorithme, celui du bijoutier, pour créer un arbre binaire et pour construire les algorithmes permettant de le parcourir. Mieux qu'un grand discours montrons la construction de l'arbre correspondant aux données précédentes 7, 3, 1, 8, 6.

Le premier élément 7 est placé à la racine de l'arbre, le deuxième 3 plus petit que 7 est placé comme son fils gauche, le troisième 1 est le fils gauche de 3, puis 8, plus grand que 7 est le fils droit de 7, et enfin 6 étant plus petit que 7 il est dans la partie sous arbre gauche de 7 mais comme il est plus grand que 3 c'est le fils droit de 3 :

(6) Certains ne comptent que le nombre maximum d'échanges qui est égal à $n(n-1)/2$, ce qui me semble une bonne approche avec les élèves.

(7) On peut trouver un certain nombre de tris d'accès facile sur le site http://prevert.upmf-grenoble.fr/Prog/Tris/tableDesMatieres.html???.html

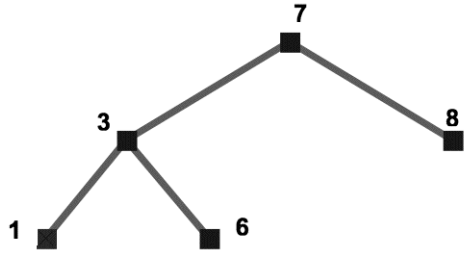

Figure 3 : Arbre binaire et tri

Nous avons bien appliqué l'algorithme du départ (du bijoutier) car les éléments « à gauche » de la racine sont plus petits que cette racine et ceux à droite plus grands ; d'autre part une projection de l'arbre sur une horizontale nous donne la liste triée.

Pour représenter cet arbre dans un tableau nous avons besoin de 4 lignes : les indices, les valeurs, les indices des fils droits et les indices des fils gauches.

Indices	1	2	3	4	5
Valeurs	7	3	1	8	6
Fils gauche (indice)	2	3	-	-	-
Fils droit (indice)	4	5	-	-	-

Figure 4 : Représentation d'un arbre binaire

La construction d'un tableau avec les indices des fils gauches et des fils droits n'est pas obligatoire ; on peut en effet travailler en gérant l'indice des fils droits et gauches[8].

Il n'y a que six façons de parcourir un arbre binaire, notons R la racine R, G le sous-arbre gauche et D le sous arbre droit : nous aurons alors les parcours possibles suivants : GRD, RGD, GDR, DGR, RDG et DRG.

RGD est appelé préfixé, GDR postfixé et GRD infixé.

Pour avoir la liste triée il suffit d'appliquer l'algorithme de parcours (infixé) suivant :

```
PROCEDURE Parcourir (Arbre)
        SI (Arbre = vide) ALORS Arrêt
        SINON  Parcourir (Gauche (Arbre))
               Traiter (Racine)
               Parcourir (Droite (Arbre))
FIN PROCEDURE Parcourir
```

Ici Traiter peut être une opération quelconque, l'impression par exemple donnera une liste triée. Le calcul de complexité n'est pas trop difficile[9], mais pour qu'elle soit « acceptable » (en O(log(n)) l'arbre doit être assez bien équilibré.

(8) Un cours concernant les arbres binaires peut être consulté sur http://cslibrary.stanford.edu/110/BinaryTrees.pdf

(9) Concernant la complexité du tri par arbre binaire : http://www.dailly.info/Tri-par-arbre-binaire-de-recherche

Notons au passage que dans les arbres binaires on retrouve les représentations des expressions algébriques et leurs diverses écritures linéaires :

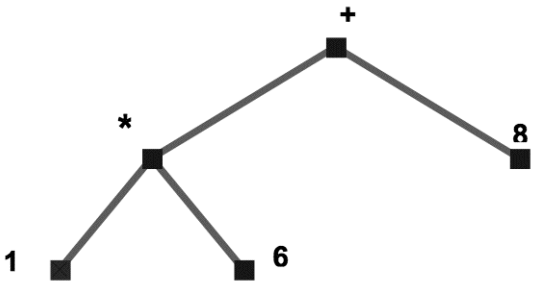

Figure 5 : Arbre binaire et expressions algébriques

qui peut être écrit + (× 1 6) 8 (notation polonaise = parcours RGD) ou bien 1 6 × 8 + (notation polonaise inverse = parcours GDR) ou encore 1 × 6 + 8 (parcours GRD). On le voit : ce travail sur les arbres binaires et les algorithmes de parcours de ces arbres(10) dépasse largement le cadre strictement informatique.

5. Des activités dans la classe

Bien entendu les activités précédentes n'ont pas pour objectif de nourrir un cours d'algorithmique mais bien plutôt de donner une démarche, un cap pour s'orienter.
Dans les cours que j'ai fait à Montréal (2 heures par semaine aux cours secondaires 4 et 5) il y a déjà quelques années, nous avions orienté les activités des élèves et donc le contenu du livre (ARCOUET M., ORIOL J-C, 1986) qui allait avec autour de deux axes :

– Premier axe : Un certain nombre de chapitres organisés autour de contenus

Cet enseignement était fondé sur des situations problèmes avec un apport théorique minimum et des problèmes plus difficiles pour les élèves à qui le B-A-BA ne suffisait pas.
J'ajoute que chaque activité des 10 chapitres donnait lieu à une analyse descendante faite, surtout au début, de réécriture de la solution dans un pseudo langage, puis de quatre codages de chaque algorithme : en Basic, en Pascal et en Logo français et en Logo anglais.
Voici un exemple simple (page 77) de l'introduction de la structure si … alors … sinon … :

> « LE CHOIX DE LA FORME : SI … ALORS … SINON …
>
> Lorsque le choix ne comporte que deux possibilités, il s'agit d'une alternative. Celle-ci peut être formulée en une seule phrase de la forme SI … ALORS … SINON … Il est préférable de choisir cette dernière formulation, plutôt que d'employer deux fois la forme SI... ALORS...

(10) Un certain nombre d'algorithmes concernant les arbres binaires peuvent être consultés sur http://cslibrary.stanford.edu/110/BinaryTrees.pdf

EXEMPLE
La note du bulletin est inférieure à 60 % AFFICHER « Échec ». La note est égale ou supérieure à 60 % AFFICHER « Succès ».
En pseudo-langage :

```
CHOIX
  SI note< 60% ALORS afficher "Échec"
  SINON afficher "Succès"
FIN du CHOIX
```

Suivent ensuite les codages dans les quatre langages cités.

– Deuxième axe : une Annexe catalogue des projets suggérés et grille d'appréciation

Voici par exemple le projet nº 90 (page 252) :

« Projet 90

La fréquence d'une lettre, d'un mot, d'une expression

L'ordinateur évalue la fréquence d'une lettre, d'un mot ou d'une expression rencontrés dans un texte donné. Demandez à l'ordinateur d'établir la liste des lettres contenues dans un texte, par ordre de fréquence. Ce petit exercice vous en apprendra beaucoup sur la fréquence de certaines lettres dans la langue française. La lettre « t » est-elle souvent suivie de la lettre « r » ? Combien y a-t-il de mots contenant trois consonnes consécutives (ex. : installer)?

Vous pourriez aussi comparer entre elles les fréquences de certaines lettres de deux langues différentes.

Par ailleurs, la fréquence d'un mot ou d'une expression dans un texte aide souvent à en préciser le sens et à mieux saisir les intentions de l'auteur. Par exemple, quelle est la fréquence des mots « amour », « neige », « pays », dans les poèmes de Gilles Vigneault? Quelles conclusions pouvez-vous en tirer?

Pour analyser ce projet, vous devez penser que l'ordinateur devra pouvoir identifier chacun des éléments (lettre, mot ou expression) d'un texte et les comparer avec l'élément recherché. Il devra aussi pouvoir compter le nombre de fois que cet élément est rencontré. »

Pour aider à évaluer les projets nous avons intégré au manuel une grille d'appréciation des projets du catalogue.

6. En guise de postface

Mes expériences d'enseignement de l'informatique et mes connaissances dans ce domaine m'ont convaincu de plusieurs points à partir desquels les pratiques sur l'algorithmique pourraient avancer :

– L'algorithmique est une science et elle s'apprend.
– La pratique de traduire les algorithmes en un pseudo code doit être préalable au codage sur une machine.

– Les preuves de programme, invariants, écriture récursive des programmes, dans les cas simples, sont tout à fait atteignables par des élèves de seconde.
– Le codage sur des ordinateurs est un élément important car il sous-tend actuellement toutes les activités scientifiques et autres.
– L'organigramme est un outil inadapté à la transcription des algorithmes ; il est dangereux car son apparente simplicité pour des boucles toutes simples se transforme en véritable « plat de spaghettis » indigeste dès que le problème se complique(11).
– Cet enseignement devrait faire partie de la formation du citoyen.

Et pour atteindre de tels objectifs il faut s'en donner les moyens, en temps pour les élèves, en formation pour les enseignants, etc., afin que l'enseignement des algorithmes ne devienne pas les chromes que l'on fait briller pour cacher le mauvais état de la machine.

Vous aviez dit « algorithme » ? Je crois que l'on peut commencer à discuter !

7. Bibliographie

ARCOUET M., ORIOL J.-C., 1986, Projets et programmes (Introduction à la science de l'informatique), ERPI, Éditions du renouveau pédagogique Inc., Montréal (Québec).

CHABERT J.-L., et coll., 1995, Histoire d'algorithmes, Belin, ISBN 2-7011-1346-6.

DOUADY R., 1986, Jeux de cadres et dialectique outil/objet, Recherches en didactique des mathématiques, Vol. 7.2, p. 5-32, La pensée sauvage éditions.

FRIGBER J., 2007, A Remarkable Collection of Babylonian Mathematical Texts Manuscripts in the Schøyen Collection : Cuneiform Texts I, 536 p., ISBN: 978-0-387-34543-7.

HOFSTADER D.R., 1980, Gödel, Escher, Bach, Vintage Books Edition.

IREM Montpellier, 2009, http://ens.math.univ-montp2.fr/SPIP/-Anthypherese-

KNUTH D., 1998, Sorting and Searching, Second Edition (Reading, Massachusetts : Addison-Wesley), ISBN 0-201-89685-0.

LÉVY G., 1994, Algorithmique combinatoire, Dunod, ISBN 2 10 002149 4.

ORIOL J.-C., SCHWARTZ C., 1981, Matchinettes 3, Irem de Grenoble.

RHOL J.S, 1984, Recursion via Pascal, Cambridge University Press, ISBN 0 521 26329 8.

RONAN C., 1988, Histoire mondiale des sciences, Seuil, ISBN 2-02-010045-2.

WAND M., 1984, Induction, Recursion and Programming, North Holland, ISBN 0-444-00322-3.

WILLIAMS J.W.J., Algorithm 232 : Heapsort. Communications of the ACM, 7, 6, 347-348 1964.

(11) Il me semble avoir écrit dans une brochure APMEP il y a longtemps un article intitulé « L'organigramme une planche savonnée », mais je ne retrouve pas la référence.