Spontanéité, créativité et algorithmique

P.A. de Marneffe
Institut Montefiore
Université de Liège, Belgique

J'aime la règle qui corrige l'émotion.
Georges Braque

0. Introduction

L'enseignement de l'algorithmique implique la mise en oeuvre d'exercices créatifs. Les exercices sont créatifs, c-à-d qu'ils ne correspondent point à des modèles d'exercice; les énoncés ne peuvent être des décalques d'énoncés *types*. L'enseignement doit porter sur une méthode de résolution effective de problèmes : capacité de scinder le problème initial en sous-problèmes, de concevoir des processus itératifs permettant d'atteindre la solution, de choisir des structures de données adéquates. L'énumération précédente ne correspond pas nécessairement à la chronologie des opérations de résolution; au contraire, ces opérations s'interpénètrent mentalement dans la conception d'un algorithme.

Notre expérience porte sur l'enseignement d'un cours d'algorithmique pendant une période de 16 années; 8 années à des étudiants ingénieurs en troisième année à l'université, pour un groupe d'environ 150 étudiants/an; 8 années à des étudiants ingénieurs en deuxième année à l'université et à des étudiants en informatique en première année à l'université, pour un groupe d'environ 350 étudiants/an. Pendant cette période, nous avons pu observer l'arrivée à l'université des premiers étudiants ayant accompli un cycle secondaire complet selon la nouvelle organisation de l'enseignement en Belgique, l'enseignement "rénové" qui applique des méthodes pédagogiques "actives" [5].

Le cours comporte deux volets : d'une part, l'exposé d'une méthodologie de conception d'algorithmes, la méthode des commandes gardées ([1], [2], [3], [4]), et son application dans la conception des algorithmes les plus fondamentaux : le tri, la recherche en table,...; d'autre part, des exercices d'application. Les réponses aux exercices remises par les étudiants sont corrigées, annotées et restituées aux étudiants. L'examen écrit porte sur un exercice (à livre ouvert); l'oral, sur la reconstitution d'un algorithme vu dans la première partie du cours. Si l'étudiant a réussi l'examen écrit, l'examen oral est facultatif.

Les résultats de l'examen écrit se dégradent. Les écarts s'accentuent : dans les cinq dernières années, le nombre d'étudiants réussissant brillamment l'examen écrit augmente, mais le nombre total des réussites décroît pour passer de la moitié du groupe à un cinquième. La difficulté majeure provient du manque de créativité : de plus en plus d'étudiants ont de grandes difficultés à élaborer une solution pour le problème posé, à l'exposer correctement en français, et à l'exprimer dans un langage formel de programmation. Trop de tentatives de solution procèdent d'une réaction spontanée à l'énoncé sans aucune discipline de mise en oeuvre.

1. Spontanéité et créativité

La *spontanéité* correspond au *caractère d'une expression directe, sans réflexion, ni calcul*. La *créativité* est une *capacité à combiner des éléments extraits d'un ensemble limité de règles en vue de parvenir à un objectif préétabli.* La créativité est normalement au centre des objectifs d'une pédagogie active. Malheureusement, par une attitude obsessionnelle de refus de la notion de "règle", la pratique d'une pédagogie active engendre souvent des étudiants incapables de dépasser le niveau d'une réponse spontanée. Apprendre par la résolution de problèmes implique que les règles de résolution des problèmes dans un domaine donné soient connues; si elles sont ignorées, seuls des problèmes triviaux sont accessibles.

La phrase du peintre Georges Braque [6] est exemplaire : "*J'aime la règle qui corrige l'émotion."* Transposée du domaine de la peinture à celui d'une discipline scientifique, elle peut devenir : *"J'aime la règle qui corrige l'intuition."* Face à l'énoncé d'un problème, les tentatives intuitives (ou spontanées) d'une idée de solution doivent être soumises et corrigées par l'ensemble des règles qui sous-tendent le domaine. Nous avons remarqué que des spécialistes des sciences de l'éducation réagissent à la phrase de Braque en attribuant à "*corrige*" la signification "*canalise*". Cette interprétation est erronée, elle ne fait qu'exprimer l'incompréhension de la valeur positive de la notion de règles. Il n'est point question de "canaliser" la création par l'obligation de respecter des règles coercitives, mais bien de rendre la création possible par les règles qui en garantiront la cohérence et l'harmonie.

Comme premier exemple de l'aspect positif de l'utilisation d'un ensemble de règles, imaginons que nous rassemblions deux groupes de jeunes enfants; au premier groupe, nous fournissons un ensemble de matériaux et outils de base (carton, planchettes, colle, clous, marteau, scie,...) en leur proposant de les utiliser en toute liberté; au deuxième groupe, nous fournissons un tas de briques et pièces Lego, et nous leur proposons également de les utiliser en toute liberté. Quel groupe tirera le plus de satisfactions de l'expérience ? Malgré les règles très strictes imposées par les gabarits standardisés et les possibilités d'assemblage limitées des briques Lego, le deuxième groupe obtiendra certainement les résultats les plus spectaculaires et les plus homogènes. Le premier groupe est limité par la difficulté de maîtriser correctement les méthodes d'assemblage à partir de matériaux "libres". Certes, la moindre réalisation des membres du premier groupe bénéficiera d'un succès d'estime; mais ni les réalisateurs, ni les spectateurs ne seront dupes du caractère rudimentaire du résultat.

Les techniques de dessin fournissent un deuxième exemple : proposez à un groupe d'enfants de dessiner des maisons, des rues, des paysages urbains. Ensuite, introduisez les règles fondamentales de la perspective (notion de points de fuite, ligne d'horizon), et incitez-les à recommencer leurs premiers dessins. Par l'application de règles systématiques, ils auront fait un saut qualitatif extrêmement important dans leur capacité de représenter la réalité qui les entoure. L'historique du développement des méthodes de la perspective doit nous convaincre qu'il est insensé d'imaginer que ces règles pourraient être aisément "redécouvertes".

Il faut admettre l'importance pratique de la connaissance d'un ensemble de règles efficaces dans un domaine, des possibilités de combinaison de ces règles et de leurs limites d'application. Le but d'une formation doit être axée dans ce sens. Cette formation, en

exerçant la créativité de premier niveau (c.-à-d. créer dans le cadre des règles), doit également faire apparaître l'importance effective de l'existence des règles. Cette approche doit introduire l'existence d'une créativité de deuxième niveau qui consiste à composer un nouvel ensemble de règles. Cette composition est d'une difficulté de loin supérieure à celle de la créativité de premier niveau. L'erreur pédagogique est de mettre d'emblée les étudiants dans une situation de tâtonnements, de "pseudo-redécouverte" des lois d'un domaine ou d'une discipline. Il en résulte qu'à l'exception des éléments les plus brillants, les étudiants ne possèdent même plus une maîtrise suffisante des domaines de base de la formation; mêmes les capacités d'expressions écrite et orale se sont considérablement affaiblies.

2. Domaines créatifs

Toutes les disciplines ne se prêtent pas d'une manière égale à une formation par la créativité de premier niveau. Rappelons que les exercices proposés doivent avoir un caractère créatif, c-à-d que l'exercice n'a pas une réponse *unique*. En fonction des règles choisies et des combinaisons appliquées, la solution peut être différente. Les étudiants doivent également disposer de critères suffisamment objectifs pour juger de la qualité d'une solution qu'ils proposent. Par critère *objectif*, nous entendons un critère lié à une qualité intrinsèque d'une solution et non en fonction d'une convention entre l'enseignant et les étudiants ("Je choisis cette solution parce qu'elle plaira au professeur.").

Les études classiques offrent une large place à ces domaines créatifs. Les exercices de traduction de textes en langues anciennes remplissent probablement le mieux toutes les conditions de la méthode créative : la traduction exige d'une part, une excellente connaissance des règles de grammaire de la langue du texte original, et d'autre part, une maîtrise de la langue maternelle pour la traduction proprement dite. Les étudiants apprennent progressivement à maîtriser un ensemble complexe de règles (la grammaire) et le large spectre des nuances qui résultent des multiples choix possibles pour la traduction. Les critères de qualité restent très objectifs : ils sont liés à la fidélité au sens du texte initial ("pas de contresens") et à l'harmonie de la traduction.

La dissertation présente les mêmes qualités. Mais, il faut garantir l'aspect objectif des critères de jugement appliqués au résultat. La manière d'argumenter doit être jugée, et non la conformité de la conclusion avec le point de vue de l'enseignant.

La créativité est plus difficile à introduire dans l'étude des disciplines scientifiques. Il faut éviter de tomber dans une systématique de l'exercice. Il faut éviter le piège de l'exercice "type" où le problème posé ne requiert pas pour sa solution la maîtrise d'une méthode, mais simplement de pouvoir distinguer à quel modèle de résolution la solution appartient. La méthode scientifique est fondamentalement très complexe : elle présuppose une inclination à l'interrogation, à l'étonnement; elle exige la maîtrise de la formulation d'hypothèses et de la conception de procédés de vérification. Les travaux de recherche de "documentation" sur un thème donné dans une optique multidisciplinaire ne sont que de piètres simulacres de la méthode.

L'algorithmique permet une approche créative. Les critères de jugement de la qualité d'une solution restent totalement objectifs : d'une part, l'exactitude de la solution, d'autre part, les performances de la solution au point de vue temps d'exécution et espace mémoire requis.

3. Le rôle de la mémorisation

Appliquer efficacement un ensemble de règles exige un travail préalable de mémorisation : il faut que les règles disponibles soient connues. Sans la connaissance immédiate d'ensemble d'informations, de propriétés associées au domaine, aucune réalisation n'est possible avec un minimum d'efficacité. Il importe de connaître l'alphabet pour manipuler efficacement les index de référence; il faut connaître les tables de multiplication pour calculer rapidement. Les ouvrages de référence (grammaire, lexique, tables, dictionnaires,...) doivent être disponibles, mais il faut connaître de mémoire un minimum pour réduire au maximum le temps de consultation et d'investigation. Par exemple, la syntaxe des langages de programmation doit être maîtrisée sans un recours constant au manuel de référence.

Remarque. Même des notions aussi décriées que la chronologie historique ont leur importance. Par exemple, si l'on sait que Christophe Colomb a atteint le continent américain en 1492 et que Charles Quint est né en 1500, on peut inférer immédiatement que le contexte de formation du jeune roi a été très différent de celui de Philippe le Beau, son père. *Fin de remarque.*

4. Les difficultés d'application de la créativité

La mise en oeuvre de la méthode se heurte à deux obstacles majeurs : les étudiants doivent accepter une approche critique de la résolution d'un problème et admettre de s'exercer sur des problèmes "gratuits".

Dans l'exposé de sa méthodologie de travail [6], le peintre Braque a une deuxième phrase révélatrice : "*Il faut toujours avoir deux idées, l'une pour détruire l'autre*". Le choix parmi les règles à appliquer exige le même état d'esprit : pourquoi choisir telle règle plutôt qu'une autre ? Cette attitude est tout à l'encontre d'un comportement spontané. Malheureusement, nous devons constater que cette attitude critique ne résulte pas nécessairement d'une propension innée chez les étudiants. Au contraire, la légitimation de comportements spontanés dans l'éducation a renforcé l'écart entre les plus doués et les autres.
Exemple. A une demande de justification d'un choix dans la conception d'un algorithme (pourquoi telle structure de donnée ? pourquoi tel algorithme de tri ?...), la réponse la plus courante est "C'est la première idée que j'ai eue". Il a y cinq ou six ans, ce type de réponse était l'exception.
Fin de l'exemple.

Une version latine est un acte "gratuit"; de même la dissertation. Dans l'acquisition d'une méthode de résolution de problèmes, la démarche doit primer sur l'aspect utilitaire des problèmes posés. Les énoncés peuvent être des abstractions de problèmes réels qui ont été débarrassés d'une multitude de détails superflus. Mais, même dans ce cas, l'aspect "gratuit"

du problème est un obstacle pour de nombreux étudiants. L'abstraction de problèmes réels est inévitable si on veut développer les compétences méthodologiques. Par exemple, en algorithmique, concevoir un programme qui implique l'emboîtement d'un processus itératif à l'intérieur d'un autre processus itératif exige une maîtrise de la méthodologie de conception de loin supérieure à ce que requiert le développement d'un programme basé sur une seule itération. Déterminer la somme des éléments d'un vecteur est plus simple que d'isoler des éléments qui respectent certaines conditions dans un tableau à deux dimensions. Ce deuxième problème est une abstraction de problèmes réels de reconnaissance de formes ou d'analyse d'images.(Voir exemple d'énoncé en annexe.)

Le contexte de l'enseignement de l'algorithmique a subi une mutation importante dans les années récentes : la multiplication des ordinateurs personnels. L'effet n'est pas nécessairement un accroissement de l'intérêt des étudiants pour l'algorithmique en tant que discipline. Beaucoup d'étudiants disposent d'une machine personnelle, mais la plupart ne possède aucun logiciel de compilation. La programmation de l'équipement ne suscite pas leur curiosité; seule la disposition de logiciels d'application les intéresse. En outre, égarés par des propos très optimistes sur les possibilités de l'Intelligence Artificielle, ils imaginent que la conception des logiciels d'application provient de la mise en oeuvre quasi automatique de systèmes de programmation appropriés. Il en résulte que l'algorithmique paraît *obsolète*, son étude devenant aussi "gratuite" que celle du latin.

5. Une approche créative en algorithmique

Les règles à combiner pour la résolution de problèmes en algorithmique doivent être des règles effectives, c-à-d plus précises et plus directives que de simples conseils. Elles doivent servir de guide pour le passage d'une intuition de solution à une solution effective.

La méthode algorithmique repose essentiellement sur la notion de récurrence. Une des règles les plus fécondes pour guider le développement d'un algorithme itératif est l'*invariant d'instruction de répétition.* La méthode n'est pas simple; elle demande une formation pour être appliquée correctement. D'autre part, elle n'est pas *évidente*; il est intéressant de savoir qu'elle n'était pas connue par Turing [7, page 145]. A notre avis, il faut introduire la méthode des invariants de boucle le plus tôt possible dans les cours d'introduction à l'algorithmique. Il faut montrer l'aide qu'elle peut apporter dans la conception raisonnée d'une solution. A titre d'exemple, nous montrerons son application au problème du "robot peleur de pommes de terre" qui est souvent introduit dans les cours d'initiation ([8],[9]). L'énoncé du problème est simple, et l'expérience a suffisamment montré que les programmeurs débutants, à leur grand étonnement, ont beaucoup de difficultés à concevoir une solution générale couvrant tous les cas initiaux possibles.

Enoncé : Le robot dispose d'un panier qu'il peut aller remplir (commande "remplis") à un tas de pommes de terre dont le contenu est inépuisable. Par la commande "pèle", le robot prend une pomme de terre dans le panier, la pèle et la jette dans la marmite. Le but est de remplir la marmite. On ne peut rien supposer sur l'état initial du panier et de la marmite, ni sur les tailles respectives de la marmite et du panier. Le robot ne peut tester que deux conditions :

"la marmite est pleine", "le panier est vide" qui peuvent être combinées par les opérateurs logiques habituels "ou", "et", "non").

Les descriptions habituelles de "marche à suivre" pour le robot font appel à une approche "opérationnelle" : le concepteur développe un programme et, par essais et erreurs, tente d'éliminer les cas non couverts. L'exactitude du programme obtenu reste une inconnue, car aucun critère ne permet de déterminer si on a envisagé tous les cas.

La méthode de l'invariant de boucle permet de guider la conception. Selon cette méthode, la commande de répétition doit être accompagnée d'une condition (l'invariant) qui est vraie avant la commande et qui reste vraie après chaque itération. A la fin de l'exécution de la commande, l'invariant et la négation du gardien de la boucle sont vrais tous les deux. La commande suit le schéma (où P est la condition invariante) :

S0; {P} **do** B-> {P **et** B} S1 {P} **od** {P **et non** B}

Une argumentation complémentaire est nécessaire pour justifier le caractère fini du nombre d'itérations.

Intuitivement, il est clair que le processus doit augmenter le nombre de pommes de terre dans la marmite sans excéder la capacité de celle-ci. Une condition invariante correspondant à cette idée de solution est d'exprimer que "la marmite contient un certain nombre de pommes de terre tel qu'elle est pleine ou non". On voit que l'opération "pèle" accroît d'une unité le nombre de pommes de terre dans la marmite. Appelons M la capacité de la marmite. Le nombre de pommes de terre présentes (à un instant donné) dans la marmite est noté *m*. La condition invariante peut être écrite : $0 \leq m \leq M$. Si $m = M$, la marmite est pleine et le problème est résolu. Vu le schéma de répétition, nous avons une proposition pour le gardien B de la commande : "la marmite n'est pas pleine" ($m < M$). En fonction des opérations de test données dans l'énoncé, le gardien devient : **non**("la marmite est pleine"). Si {P **et** B} est vrai, - dans ce cas, la condition "**non**("la marmite est pleine") et $(0 \leq m \leq M)$" est équivalente à $(0 \leq m < M)$ -, une activation de la commande "pèle" permet de passer dans un état où {P} est vrai. Initialement, la situation de la marmite est quelconque, donc la condition $(0 \leq m \leq M)$ est certainement vraie. Pouvons-nous accepter le schéma suivant pour la solution ?

{P} **do non**("la marmite est pleine") **->** {P **et** B} "pèle" {P} **od** {P **et non** B}

Non, car il faut garantir la possibilité d'effectuer la commande "pèle". Celle-ci requiert que le seau contienne au moins une pomme de terre. Il faut tester l'état du seau, et le remplir s'il est vide. Une commande de choix permet de garantir l'existence d'au moins une pomme de terre dans le seau :**if** "le panier est vide" **->** "remplis" **[] non** ("le panier est vide") **-> skip fi**
[Dans cette notation, **skip** est la commande "vide".]

Le programme complet devient :

```
{P = (0≤m≤M)}
do non("la marmite est pleine") ->
        {(P et B) = ( 0≤m<M)}
        if "le panier est vide" -> "remplis"
        [] non ("le panier est vide") -> skip
```

fi
; "pèle"
{P = (0≤*m*≤M)}
od {(P **et non** B) = (m = M)}

La terminaison est assurée. En effet, à chaque itération, le nombre de pommes de terre dans la marmite augmente d'une unité; l'expression entière (M-m) décroît à chaque itération.
[*Remarque*. Cette analyse fait apparaître une ambiguïté dans l'énoncé. En effet, il est sous-entendu que, lorsque la marmite est presque pleine, l'ajout d'une pomme de terre permet de passer de l'état d'une marmite "non pleine" à celui d'une marmite "pleine" sans qu'elle déborde, indépendamment du calibre de la pomme de terre ajoutée.]

Cette démarche de conception contient simultanément les étapes de la conception et la justification de l'algorithme développé. Tout au long de la conception, les règles guident et contrôlent l'intuition qui a servi de point de départ. Par rapport à la méthode opérationnelle, la méthode exige uniquement une certaine connaissance de la représentation formelle des conditions sous forme d'expressions booléennes (ou de prédicats).

6. Introduction à l'informatique d'application

Une démarche similaire, basée sur la compréhension d'un ensemble de règles de base, peut être appliquée dans la formation à l'utilisation des logiciels d'application. Dans le cas des traitements de texte, il est erroné de suivre une démarche d'acquisition s'appuyant sur des exemples d'utilisation du logiciel. Cette démarche aboutit à la description de séquences de touches à manipuler pour obtenir les effets escomptés. Les manuels, basés sur cette approche, deviennent plus volumineux que le texte du logiciel lui-même.

L'approche créative doit partir de la modélisation adoptée par le concepteur du logiciel. Par exemple, dans les traitements de textes, les opérations sont conçues sur base d'une "grammaire" du texte. Il est important que l'utilisateur sache que le texte est, par exemple, vu comme formé d'une séquence de paragraphes, que l'effet de certaines commandes a une portée d'application sur le texte (par exemple, une règle de justification est applicable jusqu'à la prochaine règle introduite dans le texte). La connaissance des concepts utilisés pour le développement du logiciel permet d'arriver à l'explicitation d'un mode d'emploi concis et complet.

7. Conclusions

L'algorithmique est une discipline qui se prête très bien à un enseignement basé sur une approche réellement créative. L'algorithmique pure, c.-à-d. la conception d'algorithmes, contribue à la formation générale de l'étudiant, notamment par l'apprentissage de l'utilisation de méthodes de raisonnement et d'abstraction pour généraliser la résolution des problèmes. Cette capacité d'abstraction est requise pour accéder à une créativité de "deuxième niveau".

Références

[1] Dijkstra, E.W.; *A Discipline of Programming;* Prentice-Hall Series in Automatic Computation, 1976.
[2] Dijkstra, E.W.; Feijen, W.H.J.; *A Method of Programming;* Addison Wesley, 1988.
[3] Kaldewaij, A.; *Programming : The Derivation of Algorithms;* Prentice-Hall International Series in Computer Science, 1990.
[4] Cohen, E.; *Programming in the 1990s, An Introduction to the Calculation of Programs*; Springer-Verlag, 1990.
[5] Roosen, A; *Méthodologie générale, Notes de cours.* Université de Liège, Faculté de Psychologie et des Sciences de l'Education. 1992.
[6] Prat, J.L; *Braque : Catalogue pour une rétrospective*; Fondation Pierre Gianadda, Martigny, Suisse, 1992.
[7] Wilkes, M.V.; *Memoirs of a Computer Pioneer.* MIT Press Series in the History of Computing, 1985.
[8] Dijkstra, E. W.; *EWD 316 : A short introduction to the art of programming.* THE, août 1971.
[9] Duchâteau, Ch.; *Images pour programmer.* De Boeck-Wesmael, 1990.

ANNEXE
Exemple d'un énoncé d'examen en Algorithmique
Deuxième Candidature Ingénieur, Deuxième Candidature Ingénieur Architecte.
Série A. (janvier 1994)

Soit un tableau B[0..M-1,0..N-1] de M lignes et N colonnes, avec 0<M et 0<N. Les éléments du tableau sont des entiers.

Trois éléments B[p,q], B[p,q+*k*], B[p-*k*,q+*k*], avec (*k* >0), forment les sommets d'un triangle. L'élément B[p,q] est appelé : *élément repère d'un triangle de dimension k*. Les éléments (B[p,q], B[p,q+1],..., B[p,q+(*k*-1)], B[p,q+*k*]) forment la base du triangle; les deux autres côtés sont formés par les deux séquences d'éléments (B[p,q], B[p-1,q+1],..., B[p-(*k*-1),q+(*k*-1)], B[p-*k*,q+*k*]) et (B[p,q+*k*], B[p-1,q+*k*] ,..., B[p-(*k*-1),q+*k*], B[p-*k*,q+*k*]). Si *k*>2, un ou plusieurs éléments existent à l'intérieur du triangle.

Soit B[p,q], un élément repère d'un triangle de dimension *k*. On note *s*(p,q,*k*) la somme des éléments qui forment le triangle (c.-à-d. la somme des éventuels éléments internes et des éléments qui forment les trois côtés, un élément sommet n'étant compté qu'une seule fois).

Par définition, B[i,j] est un *élément de type k* si les quatre conditions suivantes sont vérifiées:
a) B[i,j] est un élément repère d'un triangle de dimension *k*;
b) B[i,j+1] est un élément repère d'un triangle de dimension *k*;
c) B[i,j] > B[i,j+1];
d) *s*(i,j,*k*) > *s*(i,j+1,*k*).

On demande de concevoir un programme qui, recevant comme données le tableau B et une constante entière ***K*** (avec ***K***>2), détermine la valeur d'une variable entière **nbel** telle que cette valeur corresponde au nombre d'*éléments de type **K*** qui existent dans le tableau B

donné. En outre, le programme construira une liste liée permettant de repérer les éléments B[i,j] qui sont des *éléments de type **K***; chaque cellule contiendra, pour un de ces B[i,j], les indices de l'élément (i, j) et la valeur de la différence (s(i,j,***K***)-s(i,j+1,***K***)) .
S'il n'existe aucun *élément de type **K*** dans le tableau B, **nbel** aura la valeur 0 et la liste liée sera vide.