\

UNE REPRÉSENTATION DES ALGORITHMES GARDANT TRACE DE LA CONCEPTION

Ruddy LELOUCHE, Ph. D.

Département d'Informatique
UNIVERSITÉ LAVAL
Québec, CANADA G1K 7P4

Téléphone : (1-418) 656-2597 ou 656-7979
Télécopieur : (1-418) 656-2324
E-mail: LELOUCHE@IFT.ULAVAL.CA

Résumé

Rarement enseignée en tant que telle, l'algorithmique est au mieux intégrée dans l'enseignement de la programmation, et la représentation des algorithmes utilisée dans les manuels ne traduit que le résultat de leur conception. Dans cet article, nous montrons en quoi la représentation courante des algorithmes est insuffisante et comment l'algorithmique doit être surtout un outil de conception. Nous présentons ensuite une méthode générale de construction d'un algorithme pour résoudre un problème, et indiquons comment nous avons été conduit à présenter les algorithmes sous forme de blocs de développement. Enfin, nous énumérons les principaux avantages de cette technique de représentation.

INTRODUCTION

Suite aux travaux de Dijkstra [1968, 1976] et d'autres [Hoare 1969, Dahl & al. 1972, Wirth 1976, Gries 1981], la plupart des manuels enseigne maintenant à programmer de manière structurée. Pour le développement du programme, certains invitent l'étudiant à utiliser d'entrée de jeu le langage informatique qui sera utilisé [Schneider & al. 1982, Mills & al. 1987]. D'autres préconisent plutôt de représenter le programme en cours de développement indépendamment du langage cible [Cooper & al. 1985, Tremblay & al. 1985, Goldschlager & al. 1986]. Dans tous les cas, grâce à la programmation structurée, l'utilisation des algorithmes a très largement remplacé celle des organigrammes.

Souvent, l'algorithme est présenté comme une simple représentation du programme à écrire, et l'algorithmique comme une technique de développement des algorithmes et programmes. Cette vision est exacte mais insuffisante: l'algorithme peut rendre aux programmeurs, surtout débutants, de bien plus grands services. En effet, l'algorithmique est aussi et surtout une *méthode de conception* permettant d'élaborer et de développer un algorithme résolvant *correctement* le problème posé[1]. Cette méthode pourrait donc être utilisée indépendamment de l'informatique. Il est souhaitable qu'elle soit *intégrée*, qu'elle fasse référence à une démarche de résolution explicite, et surtout que *les traces de cette démarche subsistent* dans les algorithmes et programmes automatisant cette résolution.

Je montrerai d'abord en quoi la représentation la plus courante des algorithmes est insuffisante (section 1). Je présenterai ensuite une méthode générale de construction d'un algorithme permettant de résoudre un problème; j'indiquerai comment l'algorithmique est un outil de conception, et comment j'ai été conduit à présenter les algorithmes sous forme de blocs de développement (section 2). Enfin, je passerai en revue les avantages majeurs apportés par cette technique de représentation (section 3).

[1] Dans son acception la plus générale, l'algorithmique traite aussi de tout ce qui concerne la conception d'algorithmes *efficaces*; cependant, je ne m'intéresse pas ici à cet aspect, essentiellement abordé dans d'autres cours: structures de données, analyse d'algorithmes, etc. En outre, le calcul de programmes et certains algorithmes permettent d'améliorer l'efficacité des programmes sans changer leurs spécifications fonctionnelles.

\

1. UN EXEMPLE

1.1 Énoncé du problème

Soit le problème simple suivant. Un robot vient d'entrer dans un labyrinthe du type de celui indiqué figure 1, par la porte d'entrée, qui s'est hermétiquement refermée derrière lui. Il faut programmer le robot pour le faire sortir du labyrinthe. Celui-ci est naturellement supposé être plan et ne contenir qu'une région (ces hypothèses ont pour seul but de garantir l'existence d'une solution au problème). Le robot sait avancer d'un pas égal à la largeur des couloirs, et effectuer un quart de tour à gauche ou à droite. Il peut aussi tester la présence d'un mur devant lui, et tester qu'il est sorti. En fait, ce robot est très proche de celui de Richard Pattis [1981], mais beaucoup plus simple encore.

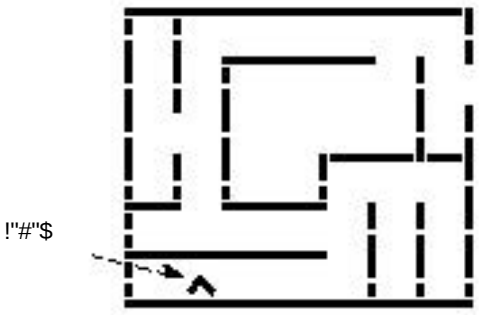

Figure 1. - Labyrinthe et robot.

1.2 Solutions "parachutées"

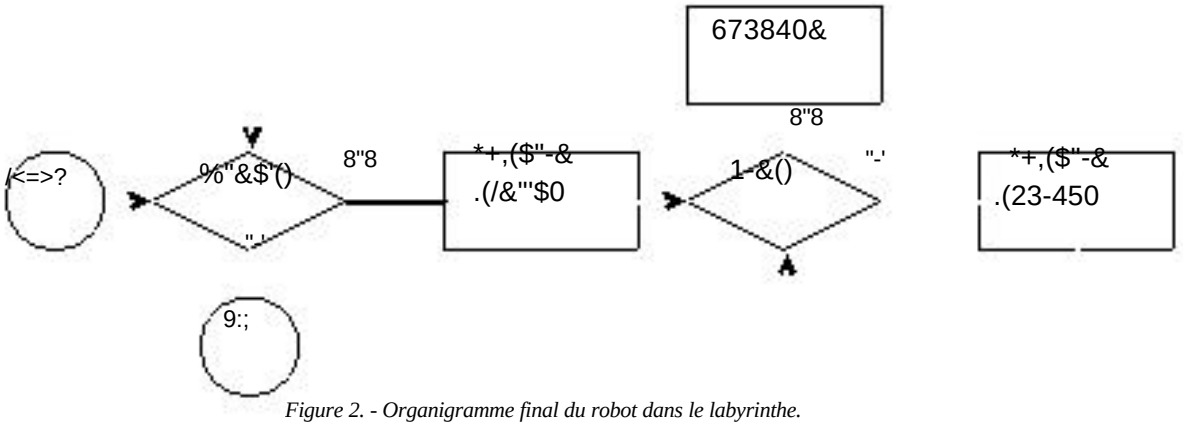

Figure 2. - Organigramme final du robot dans le labyrinthe.

J'ai été initialement confronté au problème du robot dans le labyrinthe en visionnant un film éducatif consacré aux organigrammes [Eduvision 1978]. Dans ce film, la solution était présentée en adoptant une *approche par essais et erreurs*: un organigramme intuitif était d'abord proposé, puis mis en échec sur une configuration adéquate du labyrinthe, ce qui amenait à proposer un nouvel organigramme déduit du précédent par suppression ou ajout d'une "boite" dans l'organigramme, par échange de deux boites, etc. Ce processus était réitéré jusqu'à obtention d'un organigramme correct (figure 2). Notons que le seul "critère" de correction de l'organigramme obtenu est l'absence de mise en évidence d'un contre-exemple de labyrinthe qui le ferait échouer. La même solution peut aussi être présentée sous forme d'algorithme structuré (figure 3), mais cela n'était pas fait dans le film.

\

Algorithme du robot dans le labyrinthe:

```
DEBUT
TANT QUE non sorti FAIRE
        Effectuer un quart de tour à droite.
        TANT QUE il y a un mur en face FAIRE
                Effectuer un quart de tour à gauche.
                FIN TANT QUE.
        Avancer d'un pas.
        FIN TANT QUE.
FIN.
```

Figure 3. - Algorithme structuré du robot dans le labyrinthe.

Organigramme ou algorithme, l'apprenant peut légitimement se poser des questions telles que:

(1) est-il possible d'arriver à cet algorithme autrement que par essais et erreurs?
(2) pourrait-on le "réinventer", et si oui comment?
(3) comment pourrait-on s'assurer qu'il est correct, qu'il n'existe pas de contre-exemple?
(4) existe-il d'autres solutions correctes? etc.

1.3 Inconvénients de ces présentations

Le problème est le suivant: l'algorithme qui a été élaboré (quelle que soit la méthode) est présenté tout prêt "sur un plateau"; son utilisateur ne peut alors que vérifier qu'il ne trouve pas de contre-exemple le mettant en échec, et prendre ensuite le parti (= faire le pari) d'adopter l'algorithme...

L'inconvénient de ces présentations est qu'il n'y a aucune trace de la démarche de conception ayant permis d'aboutir à l'algorithme. Il est donc impossible de le reconstituer. Il est aussi impossible d'apprendre par l'algorithme à construire d'autres algorithmes. Les manuels regorgent malheureusement d'algorithmes beaux et propres, mais "parachutés", que les étudiants apprennent par cœur...

1.4 Diagnostic de ces inconvénients et solution proposée

En fait, l'algorithme des figures 2 et 3 répond à une *stratégie* de résolution simple: *on fait longer par le robot le mur situé à sa droite.* Cela n'était même pas dit dans le film ! Cette connaissance est pourtant fondamentale, puisque c'est elle qui permet de répondre aux questions posées ci-dessus :

(1) oui: en utilisant la stratégie qui vient d'être indiquée;
(2) oui: il suffit de se souvenir de la stratégie adoptée;
(3) oui: on peut le démontrer en raisonnant à partir de l'explicitation de la stratégie et des hypothèses concernant la configuration du labyrinthe;
(4) oui: par exemple en suivant le mur de gauche.

La solution au problème de l'opacité des algorithmes est donc simple: il faut que leur représentation indique la stratégie adoptée pour arriver à leur expression finale. Certes, on peut exprimer cette stratégie en "mettant des *commentaires*". Mais cela ne suffit pas dans un cours, d'une part parce que de nombreux commentaires n'ont rien à voir avec la stratégie adoptée, et surtout parce que rien ne garantit que l'étudiant pensera à les mettre... En outre, quel est le champ d'action de chaque commentaire: l'instruction située sur la même ligne? celle qui précède ou qui suit? plusieurs instructions? Dans l'état actuel des choses, j'ai l'impression que, d'une certaine manière, les commentaires sont à la démarche de conception ce que les organigrammes sont aux algorithmes structurés...

L'algorithme du robot, tel que je le construis devant mes étudiants, et tel que je l'exige pour leurs travaux, ressemble en fin de parcours à la figure 4. La suite de ce texte expose ce qui m'a amené à proposer ce type de représentation (section 2) et les avantages que j'y vois (section 3).

\

Bloc 1: Faire évoluer le robot jusqu'à le faire sortir du labyrinthe.
Stratégie: longer le mur de droite (ou de gauche).
DEBUT
TANT QUE non sorti FAIRE
Avancer d'un pas en longeant le mur de droite (voir détails bloc 2).
FIN TANT QUE
FIN.

Bloc 2: Avancer d'un pas en longeant le mur de droite.
DEBUT
Me mettre dans la bonne direction (voir détails bloc 3 ou 3 bis).
Avancer d'un pas.
FIN.

Bloc 3: Me mettre dans la bonne direction.
Stratégie: Tourner le plus à droite possible.
DÉBUT
Effectuer un quart de tour à droite.
TANT QUE il y a un mur en face FAIRE
Effectuer un quart de tour à gauche.
FIN TANT QUE.
FIN.

Bloc 3 bis: Me mettre dans la bonne direction (alternative au bloc 3).
Stratégie: Examiner si je peux aller à droite, puis en face, puis à gauche.
DÉBUT
Effectuer un quart de tour à droite {je suis prêt à aller à droite}.
SI il y a un mur en face ALORS
Effectuer un quart de tour à gauche {je suis prêt à aller en face}.
SI il y a un mur en face ALORS
Effectuer un quart de tour à gauche {je suis prêt à aller à gauche}.
SI il y a un mur en face ALORS
Effectuer un quart de tour à gauche {je dois faire demi-tour}.
FIN SI.
FIN SI.
FIN SI.
FIN.

Figure 4. - Algorithme du robot dans le labyrinthe sous forme de blocs de développement.

2. ALGORITHMIQUE ET BLOCS DE DÉVELOPPEMENT

Cette section vise à expliciter le rôle que peut jouer l'approche algorithmique dans la résolution de problèmes, et de montrer comment la prise en compte de ce rôle m'a conduit, d'une part à expliciter la démarche de conception dans la représentation finale des algorithmes, d'autre part à modifier l'approche d'enseignement adoptée pour amener les étudiants à prendre conscience de ce rôle.

2.1 Les six activités de construction d'un programme

Lorsque le programmeur - dans notre cas, l'étudiant - doit faire résoudre un problème par un ordinateur (ce qui est souvent abrégé en "programmer"), il doit successivement (ou du moins c'est l'idéal souhaité) se livrer aux activités suivantes :

1. *préciser ou se faire préciser les spécifications* du problème à résoudre;
2. s'il y a lieu, *reformuler le problème* au moyen d'une phrase (verbe et compléments), qui sera l'"énoncé abrégé" du problème; le reste de la description est en fait le contexte, ou l'environnement, rendu implicite, toujours présent dans la solution, mais non pertinent dans tous ses détails pour l'élaboration et le raffinement de l'algorithme;
3. *élaborer une stratégie de résolution* du problème, compatible avec les contraintes imposées dans les spécifications (implicites ou explicites); c'est l'étape où l'analyste utilise sa *créativité*;

\

4. *développer un algorithme* implantant cette stratégie; ce développement se fait en descendant, par raffinements successifs; le développement est terminé quand toutes les opérations terminales à effectuer sont *élémentaires*, c'est-à-dire claires ("immédiatement exécutables") pour le destinataire de l'algorithme (humain, machine, robot, équipe de programmeurs, etc.);
5. *traduire cet algorithme en programme*, en utilisant le (ou les) langage(s) de programmation approprié(s); dans le cadre d'un cours pour débutants, le langage est unique et imposé, mais sinon le choix du langage peut être plus libre et fait alors partie des spécifications du problème;
6. *tester et mettre au point le programme*, en utilisant les outils disponibles ou imposés, intellectuels (humains) ou mécaniques.

Cette méthode, très générale, est adéquate et largement suffisante pour construire et développer des programmes de l'importance et de la taille de ceux traités en première année de programmation. Elle permet même de développer des projets plus élaborés, et ce n'est guère qu'avec l'intelligence artificielle - où par définition le problème a des spécifications floues à cause de sa complexité - que les étudiants de baccalauréat (nord-américain [2]) peuvent *parfois* avoir lieu de s'en écarter.

2.2 Conception et développement de l'algorithme

L'activité 4 est la plus importante et intéressante pour l'analyste. La méthode présentée dans mes cours, relativement originale, utilise le concept de *bloc de développement*, et invite le programmeur à expliciter sa démarche à chaque étape. *Conceptuellement, chacune de ces étapes fait appel aux activités 1 à 4,* et l'algorithme de développement d'un algorithme n'est donc autre que l'explicitation des activités 1 à 4 de la liste ci-dessus (les activités 5 et 6 sont importantes aussi; voir plus loin 2.6).

En utilisant la même représentation sous forme de blocs de développement, cela est mieux explicité figure 5. La sous-activité "effectuer le développement" traduit la conception et l'écriture d'un bloc de développement. Le premier bloc de développement, premier niveau de détail du problème initial à résoudre, est en fait un algorithme résolvant ce problème; cet algorithme est *complet*, mais en général pas suffisamment *détaillé* (ce qui justifie la troisième sous-activité, "terminer le développement").

En pratique, les activités 5 et 6 sont effectuées en parallèle, en suivant une approche en remontant, pour terminer par un programme qui est la traduction exécutable de l'algorithme développé dans l'activité 4, lequel automatise la stratégie de résolution élaborée dans l'activité 3 (voir plus loin 2.6).

Bloc 1: Développer un algorithme résolvant un problème donné.
DÉBUT
Préparer le développement (voir détails bloc 2).
Effectuer le développement (voir détails bloc 3).

[2] C'est un diplôme concrétisant un premier cycle universitaire, qui équivaut à Bacc. (français) + 2 ans.