Juin 2017

Qu'est-ce-qu'un algorithme ?

Le but de ce document est de donner quelques points de repères sur la notion d'algorithme et son utilisation avec les élèves. Il est adapté à partir d'une partie du texte intitulé « Algorithmique et programmation au cycle 4 : Commentaires et recommandations du groupe Informatique de la CII Lycée », à destination d'un public d'enseignants du premier et du second degré. Dans un premier temps, nous définissons le mot algorithme, puis nous différencions un algorithme générique et un algorithme instancié. Ensuite, nous évoquons les liens entre algorithmes et programmes, et la façon d'écrire les algorithmes. Enfin, nous montrons que tous les algorithmes sont construits à partir d'un petit nombre de structures de contrôle et d'instructions élémentaires. Pour les exemples, nous nous appuyons sur l'activité « Le crêpier ».

1 Définition.

Le terme algorithme est de plus en plus fréquemment utilisé dans la vie courante, en général dans le sens de méthode, de recette, ou encore de mode d'emploi détaillé pour réaliser une certaine tâche. Pour aborder cette notion avec un objectif pédagogique, nous devons préciser certains aspects de l'utilisation de ce terme en informatique.
Sans avoir l'intention d'entrer dans des aspects trop techniques, examinons la définition proposée par Simon Modeste dans sa thèse (`https://tel.archives-ouvertes.fr/tel-00783294/file/Modeste-these-TEL.pdf`) :

> « Un algorithme est une procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes [...], la réponse au problème pour toute instance de cette famille. »

Précisons quelques-uns des termes employés :

- En informatique, un *problème* est constitué d'un ensemble d'instances (pouvant être décrites par plusieurs paramètres) et d'une question portant sur ces instances.
- Une *instance* d'un problème est obtenue en spécifiant des valeurs précises pour tous les paramètres. Les valeurs des paramètres qui constituent une instance sont aussi appelées *données d'entrée* ou simplement *entrées*.
- La réponse obtenue à l'issue de l'exécution est aussi appelée *sortie de l'algorithme*.
- Toute exécution de l'algorithme doit se terminer (*un nombre fini d'étapes*) en donnant un résultat correct (*la réponse au problème*).

Exemple.
Nous présentons ci-dessous un des problèmes les plus classiques en algorithmique, auquel se rattache l'activité « Le crêpier ».

- Le problème du tri.
- L'ensemble des instances est l'ensemble des listes finies de nombres entiers.
- Une instance particulière est par exemple la liste 5 ; 8 ; 1 ; 25 ; 46 ; 7 ; 9.
- La question est : « Quelle est la liste ordonnée dans l'ordre croissant des éléments de la liste donnée en entrée ? ».
- La sortie correspondant à l'instance 5 ; 8 ; 1 ; 25 ; 46 ; 7 ; 9 est la liste 1 ; 5 ; 7 ; 8 ; 9 ; 25 ; 46.
- Il existe de nombreux algorithmes très différents les uns des autres pour résoudre le problème du tri. Parmi les plus connus, citons le tri à bulles, le tri par sélection, le tri par insertion, le tri fusion, le tri rapide, le tri par tas, etc.
  (voir `https://fr.wikipedia.org/wiki/Algorithme_de_tri`). L'algorithme sur lequel est basée l'activité « Le crêpier » s'appelle le tri par retournement de préfixe, il ne fait pas partie des plus classiques.

Soulignons encore une fois que, quand nous parlons de *résoudre un problème*, il ne s'agit pas de le résoudre pour une instance particulière comme par exemple de trier la liste de nombres 5 ; 8 ; 1 ; 25 ; 46 ; 7 ; 9, autrement dit de répondre 1 ; 5 ; 7 ; 8 ; 9 ; 25 ; 46. Il s'agit de le traiter d'une manière générale, c'est-à-dire pour toutes les instances du problème. Par exemple : trouver un algorithme qui permet de trier n'importe quelle liste de nombres.

2 Algorithmes instanciés.

Comme nous l'avons vu ci-dessus, un point clé de la définition de la notion d'algorithme est la « généricité », c'est-à-dire le fait qu'il n'est pas destiné à répondre à une question portant sur une instance particulière, mais à toute une famille de questions similaires portant chacune sur une des instances du problème.
Dans le cas contraire, Simon Modeste parle d'algorithme « instancié ». En Scratch, il peut s'agir d'un programme (non interactif) faisant sortir le lutin du labyrinthe dessiné sur l'arrière-plan. Bien entendu, ce programme ne permettra pas de faire sortir le lutin d'un autre labyrinthe. Il existe plusieurs algorithmes permettant de sortir de n'importe quel labyrinthe, comme par exemple l'algorithme de Pledge (`https://interstices.info/jcms/c_46065/l-algorithme-de-pledge`), mais la difficulté de cet algorithme est bien plus grande.

La distinction entre algorithme et algorithme instancié est importante pour l'enseignant. En effet, parmi les premiers exemples d'algorithmes rencontrés par les élèves, il y aura en réalité de nombreux algorithmes instanciés, comme par exemples les algorithmes de déplacement du type « Robot idiot », la plupart des exercices de tracé de figure, ou encore les diverses animations réalisées en Scratch.

Concrètement, les programmes de calcul sont des exemples d'algorithmes non instanciés qui se rencontrent assez souvent. Ils sont intéressants du point de vue de la généricité, mais ils ont le défaut d'être en général très pauvres d'un point de vue algorithmique. En effet, ils consistent le plus souvent uniquement en une séquence d'instructions, et ne comportent aucune des structures de contrôle (branchements et boucles) qui font la richesse de l'algorithmique.

D'autre part, les manuels proposent aussi des algorithmes non instanciés utilisant des variables pour lesquelles une valeur est demandée à l'utilisateur, comme la longueur du côté d'un carré à tracer. Un exercice de ce type est intéressant et particulièrement important car il permet d'introduire les variables informatiques et le fait qu'elles sont presque indispensables pour apporter une généricité.

Enfin, il peut également arriver qu'un algorithme non instancié soit attendu, alors même que le problème est présenté à partir d'un exemple, c'est-à-dire d'une instance du problème. C'est un point qui prête très souvent à confusion pour les élèves, et auquel il est nécessaire de porter attention.
Ainsi, dans l'activité du crêpier, les élèves commencent par travailler à partir d'une certaine pile de crêpes. Le problème est donc instancié au départ, mais la production attendue est bien un algorithme générique, c'est-à-dire non instancié, qui permettra de trier n'importe quelle pile de crêpes. Il est donc nécessaire, à un moment ou à un autre, que l'enseignant précise que ce qui est attendu, c'est bien une méthode qui marche pour n'importe quel état de départ de la pile de crêpes.

3 Faut-il distinguer algorithme et programme ?

Les deux notions sont bien entendu proches, mais notons qu'un programme n'est pas exactement un algorithme. Plus précisément un programme est la mise en œuvre d'un ou plusieurs algorithmes dans un langage de programmation donné, et cela fait intervenir beaucoup d'autres notions (syntaxe du langage, techniques de programmation, etc.). Une autre façon de le dire est de noter qu'un programme est destiné à être exécuté par une machine, alors qu'un algorithme est destiné à être compris par un être humain.

La plupart des manuels de mathématiques de cycle 4 différencient les deux notions, à partir de cette dichotomie humain/ordinateur. Dans les derniers programmes scolaires de mathématiques de cycle 4, ainsi que dans les documents d'accompagnement, la distinction entre les deux est également faite.
Le document d'accompagnement (page 3) commente ainsi :

> « Le programme définit comme attendu de fin de cycle : « *Écrire, mettre au point et exécuter un programme simple.* » Il met l'accent sur l'analyse d'un problème en vue de concevoir un algorithme, mais aussi sur la phase de programmation effective, indissociable d'une étape de mise au point. Cependant, il ne faut pas nécessairement insister sur une chronologie précise, ces trois phases avançant souvent en parallèle : la mise au point comprend la phase d'essais-erreurs où l'élève construit petit à petit un programme qui répond au problème qu'il s'est posé. De même, au moment de la programmation effective l'élève peut se retrouver confronté à une question algorithmique qu'il n'avait pas prise en compte dès le départ. »

Il nous semble également intéressant de commencer à distinguer dès le début le raisonnement algorithmique de l'activité de programmation. La distinction entre algorithme et programme devient encore plus importante au lycée, et bien entendu dans les études supérieures d'informatique.

L'activité du crêpier est ainsi une occasion de se concentrer sur l'aspect algorithmique, indépendamment de tout objectif de programmation. Les activités sans ordinateur font partie des préconisations

du document d'accompagnement (page 2) :

> « Les modalités de l'apprentissage correspondant peuvent être variées : travail en mode débranché, c'est-à-dire sans utilisation d'un dispositif informatique, individuel ou en groupe, en salle informatique ou en salle banale, sur tablette ou sur ordinateur. »

4 Écriture d'algorithmes

4.1 Sous quelle forme écrire un algorithme ?

Comme nous l'avons vu précédemment, la personne qui écrit un algorithme s'adresse à un être humain : elle-même, un peu plus tard, quand elle écrira un programme mettant en œuvre cet algorithme, ou bien quelqu'un d'autre. C'est pourquoi l'écriture d'un algorithme doit faire apparaître au mieux son fonctionnement et sa structure : les entrées, l'enchaînement des instructions et les structures de contrôle, et enfin la sortie. Autrement dit, l'important dans l'écriture d'un algorithme, c'est de faire en sorte que l'aspect sémantique (le sens) se trouve le plus possible mis en évidence et dissocié de l'aspect syntaxique (la forme) lié aux langages de programmation. Nous présentons plusieurs possibilités dans ce paragraphe, sans pour autant préconiser l'utilisation exclusive de l'une d'entre elles, bien au contraire !

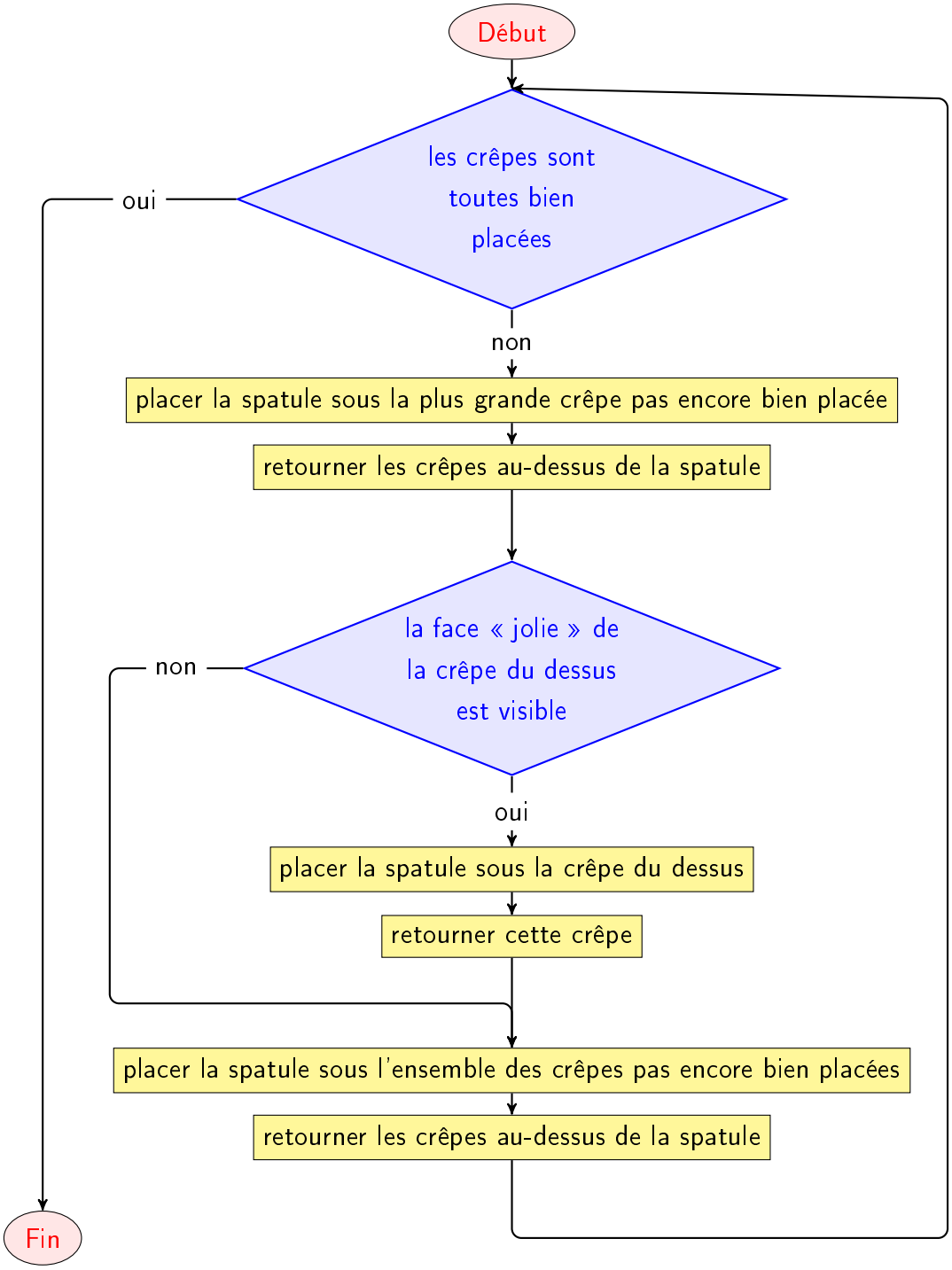
Très souvent, les algorithmes sont écrits en « pseudo-code », c'est-à-dire sous une forme proche de celle d'un programme, en simplifiant un peu les contraintes syntaxiques. De nombreuses variantes sont possibles, nous proposons deux exemples ci-dessous.

```
début
    répéter
        placer la spatule sous la plus grande crêpe pas encore bien placée
        retourner les crêpes au-dessus de la spatule
        si la face « jolie » de la crêpe du dessus est visible alors
            placer la spatule sous la crêpe du dessus
            retourner cette crêpe
        placer la spatule sous l'ensemble des crêpes pas encore bien placées
        retourner les crêpes au-dessus de la spatule
    jusqu'à ce que les crêpes soient toutes bien placées
fin
```

```
begin algorithm {
   repeat until { les crêpes sont toutes bien placées }
   do {
      placer la spatule sous la plus grande crêpe pas encore bien placée ;
      retourner les crêpes au-dessus de la spatule ;
      if { la face « jolie » de la crêpe du dessus est visible }
      do {
         placer la spatule sous la crêpe du dessus ;
         retourner cette crêpe ;
      } end if
      placer la spatule sous l'ensemble des crêpes pas encore bien placées ;
      retourner les crêpes au-dessus de la spatule ;
   } end repeat
} end algorithm
```

L'utilisation d'un pseudo-code a pour avantage de simplifier le passage à la programmation, du moins si les conventions d'écriture utilisées sont proches de celles du langage de programmation visé. Cependant, cette façon de faire tend à introduire un langage relativement formalisé pour écrire les algorithmes. Une telle contrainte est quelque peu en contradiction avec la notion d'algorithme vu comme une méthode abstraite de résolution d'un problème, qui peut donc être mis en œuvre dans n'importe quel langage de programmation (ou pas). Par ailleurs, notons que le pseudo-code exige un minimum d'apprentissage. Finalement, retenons que le pseudo-code unifie d'une manière simple l'écriture des algorithmes mais que son danger est que des éléments de programmation et d'algorithmique sont souvent mélangés.

Il n'y a pas que le pseudo-code qui permet de dissocier (au moins relativement) le sens de la forme. Toute écriture d'un algorithme pas trop contrainte par une syntaxe particulière le permet. Par exemple, des schémas appelés « organigrammes », « logigrammes » ou « algorigrammes », qui permettent de représenter des algorithmes simples à l'aide de quelques conventions graphiques, sont parfois utilisés. Ces représentations graphiques peuvent être très formalisées : il existe une norme ISO 5807 régissant leur écriture. Elles sont très répandues surtout dans les milieux industriels (et parmi les enseignants de technologie au collège et de sciences industrielles au lycée), mais très peu utilisées par les informaticiens. Il existe des logiciels graphiques spécifiques pour les dessiner. Nous proposons ci-dessous un exemple qui ne respecte pas l'ensemble des conventions graphiques de la norme ISO 5807.



L'utilisation d'une représentation graphique a pour avantage de permettre de visualiser facilement le déroulement de l'algorithme, du moins tant qu'il reste très court. Cependant, si l'objectif final est d'écrire un programme, le schéma obtenu en est encore assez éloigné, en particulier pour la gestion des boucles.

Finalement, pour écrire un algorithme sans être limité par des contraintes de forme, il est possible de simplement l'écrire avec des phrases, comme par exemple pour une démonstration en mathématiques. Il y a évidemment de multiples formulations possibles, nous proposons ci-dessous une possibilité.

> Pour commencer, on place la spatule sous la plus grande crêpe, et on retourne les crêpes au-dessus de la spatule. Si la face « jolie » de la crêpe du dessus est visible, on place la spatule sous la crêpe du dessus, et on retourne cette crêpe. Ensuite, on place la spatule sous l'ensemble des crêpes, et on les retourne. Puis on ne s'occupe plus de la plus grande crêpe qui est à sa place tout en bas, et on recommence avec les crêpes qui restent, jusqu'à ce que toutes les crêpes soient bien placées. Et c'est fini.

Du point de vue pédagogique, cette façon de faire a cependant le défaut de rendre plus difficile la mise en évidence de la structure de l'algorithme. Par exemple, dans l'activité du crêpier, si les élèves utilisent des formulations différentes pour chaque étape, il sera plus difficile de les amener à reconnaitre la boucle. À partir d'une rédaction de ce type, l'enseignant peut engager un travail pour faire identifier les différentes parties de l'algorithme, puis les faire mettre en évidence grâce à la mise en forme (passage à la ligne, indentations, etc.).

Pour conclure, il n'y a pas un « langage algorithmique » unique et il nous semble particulièrement important de confronter les élèves à une variété d'écritures d'algorithmes et de les habituer à lire et écrire des algorithmes sous différentes formes, en utilisant des conventions variées. Soulignons encore que l'essentiel est d'amener progressivement les élèves à faire la différence entre la conception et l'étude d'un algorithme et sa programmation effective. Les activités débranchées permettent de travailler ce point.

4.2 Et le crêpier ?

D'un point de vue pédagogique, les activités algorithmiques sont de plusieurs types, comme par exemple lire ou exécuter un algorithme donné, modifier un algorithme connu pour répondre à une variante du problème initial, corriger un algorithme supposé répondre à un problème donné, déterminer à quel problème connu répond un algorithme donné, et enfin concevoir et écrire un algorithme répondant à un problème. Ce dernier type d'activité est de loin le plus fondamental en informatique, mais aussi le plus difficile. C'est pourquoi il nous semble particulièrement important de faire pratiquer toutes ces variantes par les élèves de tous niveaux.

Justement, l'objet de l'activité sur le crêpier est d'amener progressivement les élèves à formuler un algorithme permettant de ranger une pile de crêpes par ordre de taille, c'est-à-dire un algorithme de tri. Cet objectif est ambitieux et c'est pour cette raison qu'il est décomposé en quatre étapes.

1. Il est d'abord demandé aux élèves d'expérimenter pour les amener à inventer et pratiquer un algorithme sans pour autant mettre en mots les actions qu'ils réalisent. Ils manipulent plusieurs fois, à partir d'états de départ différents de la pile de crêpes, jusqu'à arriver à une méthode qui fonctionne à chaque fois. Les trois autres étapes de l'activité ont pour objet d'arriver à une formulation écrite de l'algorithme implicite qu'ils ont inventé.

2. Pour la deuxième étape, il est demandé à un élève de dire à un autre élève quelles sont les manipulations à effectuer. À cette étape, des expressions comme « Mets la spatule sous cette

crêpe » seront spontanément utilisées par les enfants, en pointant du doigt. L'enseignant peut commencer à suggérer l'utilisation d'un vocabulaire plus précis, comme par exemple « les crêpes déjà triées/pas encore triées », « la plus grande crêpe », « la troisième crêpe », etc.

3. Lors de la troisième étape, celui qui donne les consignes à l'élève qui manipule ne voit pas la pile de crêpes, cachée derrière un paravent en carton. Pendant cette phase, l'utilisation du vocabulaire précédent devient plus importante. Par ailleurs, commence à se poser la question de l'arrêt de l'algorithme. Comment sait-on que c'est terminé ? Le plus souvent, l'élève manipulateur s'arrête de lui-même et dit que c'est fini, mais l'enseignant peut faire remarquer que ce n'est pas le cas d'une machine. Que se passe-t-il si on continue de manipuler alors que la pile est déjà triée ? Le plus souvent, les élèves ne s'aperçoivent pas spontanément que les manipulations inutiles ne sont pas nuisibles au résultat final, mais l'enseignant peut le faire remarquer. Il est possible aussi d'autoriser l'élève qui manipule à répondre à la question « Est-ce-que c'est terminé ? ». Attention à ne pas laisser les élèves utiliser des conditions d'arrêt erronées, comme par exemple « . . . jusqu'à ce que la plus petite crêpe soit sur le dessus ».
   Par ailleurs, c'est parfois à ce stade que la notion de répétition commence à apparaître.

4. La consigne pour la dernière partie de l'activité est de rédiger un algorithme générique que d'autres élèves pourront exécuter pour trier n'importe quelle pile de crêpes. À ce stade, l'utilisation d'un vocabulaire précis et relativement standardisé permet de faire émerger la notion de répétition, qui se traduit sous forme de boucle dans l'algorithme, pour raccourcir la formulation. Il est important d'utiliser au moins six crêpes pour s'assurer que le besoin de boucle apparaîtra.
   Dans la fiche d'activité, l'algorithme est rédigé avec des phrases, ce qui nous semble le plus adapté dans le premier degré. Pour des élèves plus âgés ou qui ont déjà une certaine pratique, il est bien entendu possible de présenter une version en pseudo-code ou un schéma.

Enfin, il est possible de prolonger l'activité pour faire utiliser une structure conditionnelle, en demandant que les faces brûlées (« pas jolies ») des crêpes soient toutes vers le bas à la fin de l'exécution.

5 Les composants des algorithmes

Les algorithmes les plus simples comme les plus élaborés peuvent tous être écrits à partir de quelques instructions élémentaires et de quelques structures de contrôle (la séquence, le branchement, la boucle).
Ainsi, un des objectifs principaux de l'enseignement de l'algorithmique est de faire en sorte que, à terme, les élèves (et les étudiants !) identifient ces composants des algorithmes quand ils les rencontrent, et sachent les utiliser à bon escient.

5.1 Instructions élémentaires

Dans l'activité du crêpier, des instructions comme « Retourner la pile de crêpes au dessus de la spatule » et « Placer la spatule sous la plus grande crêpe de la partie non triée de la pile » (ou

d'autres formulations semblables) sont des instructions élémentaires de l'algorithme : elles ne sont pas décomposées.
D'une façon générale en algorithmique, les instructions élémentaires concernent la manipulation des données. Il s'agit par exemple de l'affectation des variables, de l'évaluation des expressions, etc.

Pour faire de l'algorithmique sur papier, chacun est libre de choisir les instructions élémentaires qui lui paraissent les mieux adaptées. Ainsi, pour l'activité « Le crêpier », un jeu d'instructions élémentaires naturelles peut être *"placer la spatule sous une crêpe donnée"* et *"retourner les crêpes au-dessus de la spatule"*. Au contraire, au moment de la programmation, il y a beaucoup moins de liberté, puisque les instructions élémentaires sont forcément choisies parmi celles proposées par le langage de programmation utilisé. Par exemple, pour programmer réellement un robot qui trie une pile de crêpes, il est possible que l'instruction *"placer la spatule sous la plus grande crêpe pas encore rangée"* doive être décomposée. En effet, certains langages de programmation ne contiennent pas comme instruction élémentaire la détermination du maximum ou du minimum d'une liste de valeurs. Dans ce cas, le programmeur serait obligé de programmer lui-même un algorithme de recherche du maximum pour arriver à ses fins.

5.2 Séquences d'instructions

Il s'agit tout simplement de spécifier dans quel ordre une série d'instructions doit être exécutée. Le plus souvent, on se contente de les écrire les unes après les autres dans l'ordre d'exécution souhaité, en veillant à bien identifier le début et la fin de chacune d'entre elles.

5.3 Structures conditionnelles (« branchements »)

Les structures conditionnelles sont des structures de contrôle permettant de décrire un comportement différent de l'algorithme selon qu'une certaine condition est ou non vérifée.
Elle se présente généralement sous la forme *« si ... condition... alors ... bloc 1... sinon ... bloc 2... »*, où *condition* représente la condition à tester, *bloc 1* représente un bloc d'instructions à effectuer dans le cas où la condition est vérifiée et *bloc 2* représente le bloc d'instructions à effectuer dans le cas où la condition n'est pas vérifiée.
Il peut arriver que la partie *sinon* n'apparaisse pas : dans ce cas, si la condition n'est pas vérifiée, l'algorithme passe directement à l'instruction suivante sans rien faire.
La condition est une expression qui doit toujours pouvoir être évaluée comme vérifiée (vraie) ou non vérifiée (fausse), comme par exemple un test d'égalité de deux expressions. Dans le cas du crêpier, la condition peut-être par exemple *« la face « jolie » de la crêpe du dessus est visible »*. Si c'est vrai, il faut retourner la crêpe du dessus avant de continuer l'algorithme, si c'est faux, il n'y a rien à faire.

5.4 Structures itératives (« boucles »)

Les boucles sont des structures de contrôle permettant de décrire la répétition d'une séquence d'instructions. Elles sont de plusieurs types, voici quelques exemples :

- *« répéter n fois ... bloc ... »*. Ici, n peut être un nombre concret comme 4, ou bien une variable à laquelle une valeur numérique, éventuellement obtenue à l'issue d'un calcul, a été affectée au préalable dans l'algorithme, et *bloc* représente la séquence d'instructions à répéter.
- *« pour i allant de 1 à n, faire ... bloc ... »*. Ici, l'entier i est un *indice de boucle* qui augmente automatiquement de 1 à chaque passage dans la boucle, en commençant à 1 et en allant jusqu'à n. Cet indice i peut être utilisé dans les instructions du *bloc* formant le corps de la boucle.
- *« tant que ... condition ... répéter ... bloc ... »*. Ici, le *bloc* d'instructions sera répété tant que la *condition* est vérifiée. Si la condition n'est pas vérifiée au départ, le bloc d'instructions n'est pas exécuté du tout. Si la condition est vérifiée au départ, le bloc d'instructions est exécuté au moins une fois, et la condition est testée de nouveau à chaque sortie de la boucle, pour savoir si c'est terminé. Ce type de boucle est un peu plus délicat à manier que les précédents car il est possible d'écrire involontairement une boucle qui ne se termine jamais. C'est par exemple le cas si les paramètres qui déterminent la valeur de vérité de la condition ne sont pas modifiés dans le corps de la boucle.
- *« jusqu'à ... condition ... répéter ... bloc ... »*. Notons qu'il est possible de passer d'une boucle *tant que* à une boucle *jusqu'à* en prenant la négation de la *condition*. Il s'agit d'un exercice qui n'est pas toujours facile pour les élèves.

Pour trier 6 crêpes, les divers types de boucles peuvent être utilisés. Nous avons déjà donné une version avec une boucle *jusqu'à*, en voici deux avec des boucles *répéter* et *tant que*.

```
début
    répéter 5 fois
        placer la spatule sous la plus grande crêpe pas encore bien placée
        retourner les crêpes au-dessus de la spatule
        si la face « jolie » de la crêpe du dessus est visible alors
            placer la spatule sous la crêpe du dessus
            retourner cette crêpe
        placer la spatule sous l'ensemble des crêpes pas encore bien placées
        retourner les crêpes au-dessus de la spatule
fin
```

```
début
    tant que les crêpes ne sont pas toutes bien placées faire
        placer la spatule sous la plus grande crêpe pas encore bien placée
        retourner les crêpes au-dessus de la spatule
        si la face « jolie » de la crêpe du dessus est visible alors
            placer la spatule sous la crêpe du dessus
            retourner cette crêpe
        placer la spatule sous l'ensemble des crêpes pas encore bien placées
        retourner les crêpes au-dessus de la spatule
fin
```