L'algorithmique [al]
Support de Cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 12 mai 2018

Table des matières

L'algorithmique

Mots-Clés Algorithmique, Notion de problème, Notion d'algorithme, Algorithmes informatiques, Formuler un algorithme, Structures fondamentales d'un algorithme, Programmation modulaire, Méthode descendante ■
Difficulté ● ○ ○

Introduction
« L'algorithmique est le permis de conduire de l'informatique. Sans elle, il n'est pas concevable d'exploiter sans risque un ordinateur. » [Cormen-AL1, partie V]

Ce module introduit la notion de problème puis celle d'algorithme, précise comment le formuler, définit les structures fondamentales d'un algorithme puis décrit la programmation modulaire. L'exemple illustre la méthodologie de l'analyse descendante.

Remarque
La lecture de ce module n'est pas indispensable mais elle permet d'introduire, d'illustrer et de justifier les grands concepts qui seront illustrés dans ce cours. Elle prépare ainsi le lecteur à mieux comprendre les modules qui suivent.

1 Notion de problème

Utilité de l'ordinateur

L'ordinateur est une **machine**. Mais une machine intéressante dans la mesure où elle est destinée à nous décharger de tâches peu valorisantes et/ou rébarbatives, mais surtout parce qu'elle est capable de nous aider, voire nous remplacer, dans des tâches qu'il nous serait impossible de résoudre sans son existence (conquête spatiale, prévisions météorologiques, jeux vidéo...).

En première approche, nous pourrions dire que l'ordinateur est destiné à faire à notre place (plus rapidement et probablement avec moins d'erreurs) un travail nécessaire à la résolution de **problèmes** auxquels nous devons faire face.

Poser le problème

Un préalable, à l'activité de résolution d'un problème, est de **définir** quel est le problème posé. Par exemple, faire un baba au rhum, résoudre une équation mathématique, réussir une année d'études...

Un problème bien posé doit également mentionner l'**objectif à atteindre** (= le résultat attendu).

Enfin la formulation d'un problème ne serait pas complète sans la connaissance **du cadre dans lequel se pose le problème :**

- De quoi dispose-t-on ?
- Quelles sont les hypothèses ?
- Quelle est la situation de départ ?

Faire un baba au rhum est un problème tout à fait différent s'il faut le faire en plein désert ou dans une cuisine super équipée !

D'ailleurs, dans certains cas, la première phase consiste à mettre à sa disposition les éléments nécessaires à sa résolution : dans notre exemple, ce serait se procurer les ingrédients et les ustensiles de cuisine.

Conclusion

Un problème ne sera véritablement bien spécifié que s'il s'inscrit dans le schéma suivant :

étant donné [les données] **on demande** [l'objectif]

Parfois, la première étape dans la résolution d'un problème est de préciser ce problème à partir d'un énoncé flou : il ne s'agit pas nécessairement d'un travail facile !

2 Notion d'algorithme

Une fois le problème correctement posé, on passe à la recherche et la description d'une **méthode de résolution**, afin de savoir comment faire pour atteindre l'objectif demandé à partir de ce qui est donné. Le **nom** donné à une méthode de résolution varie en fonction du cadre dans lequel se pose le problème : *façon de procéder, mode d'emploi, marche à suivre, guide, patron, modèle, recette de cuisine, méthode ou plan de travail, algorithme mathématique, programme, directives d'utilisation...* D'où les définitions,

L'algorithmique
(Ou **algorithmie**, les deux sont autorisés) C'est *exprimer*, à une personne ou à une machine qui ne sait faire qu'un nombre d'opérations limité, *comment aboutir à un résultat* (Forêt noire, polynômes, pull irlandais, texte scientifique, etc.) *en* lui *indiquant les étapes* nécessaires.

Exemples dans la vie courante
Vous la pratiquez quotidiennement et depuis longtemps...

Données	Étapes	Résultats
Briques de LEGO	suite de dessins	Camion de pompiers
Meuble en kit	notice de montage	Cuisine équipée
Cafetière	instructions	Expresso
Laine	modèle	Pull irlandais
Farine, œufs, sucre, fruit	recette de cuisine	Tarte aux fruits
Deux nombres	multiplication	Produit des nombres

On en trouve beaucoup d'autres : mode d'emploi d'un GSM, description d'un itinéraire, etc. Il est clair qu'il y a une infinité de rédactions possibles de ces différentes marches à suivre. Certaines pourraient être plus précises que d'autres, d'autres par contre pourraient s'avérer exagérément explicatives.

Algorithme
Spécification d'un schéma de calcul sous forme d'une suite (finie) d'opérations élémentaires obéissant à un enchaînement déterminé pour parvenir à un résultat.
[Source : *Encyclopedia Universialis*]

Remarque
Le mot *calcul* doit être pris au sens large : tout ce que la machine ou la personne qui va devoir appliquer l'algorithme est capable de faire de manière automatique. L'expression principale est « résultat final déterminé » : pourvu que l'algorithme soit exact, il conduit de manière certaine et en un temps fini au résultat escompté.

Exemple : Une recette de cuisine

C'est un algorithme : elle conduit à un résultat certain pourvu que la cuisinière est capable d'effectuer chacune des opérations élémentaires décrites dans la recette à partir des ingrédients : casser un oeuf, mélanger la farine et le sucre, etc.

Exemple : La multiplication de nombres

C'est aussi un algorithme : cette méthode conduit au bon résultat pourvu que l'exécutant connaisse les tables de multiplication pour tous les chiffres de 1 à 9 et sache faire des additions.

Caractéristiques d'un algorithme

Tout algorithme possède :

- Un **nom** et s'exprime dans un **langage** (français, anglais, dessins...).
- Des **données** (farine, oeuf, sucre et fruits pour la recette ; les deux nombres à multiplier pour la multiplication).
- Un ou des **résultats** (la tarte pour la recette ; le résultat pour la multiplication).
- Une **série chronologique** d'étapes (parfois numérotées) ou **sous-algorithmes** lesquels agissent sur les données (faire la pâte, garnir la pâte, faire cuire la tarte pour la recette ; multiplier le nombre du haut par chacun des chiffres du bas, additionner tous les résultats pour la multiplication).
- Certaines phrases justifient ou expliquent ce qui se passe : ce sont des **commentaires**.

Opération bien définie

Dans certaines marches à suivre, il arrive de trouver des opérations qui peuvent dépendre de l'appréciation de l'exécutant. Par exemple, dans une recette de cuisine on pourrait lire : *ajouter **une pincée** de vinaigre, saler et poivrer **à volonté**, laisser cuire une **bonne** heure dans un four **bien** chaud, etc.*

Des instructions floues de ce genre sont dangereuses à faire figurer dans une bonne marche à suivre car elles font appel à une appréciation arbitraire de l'exécutant. Le résultat obtenu risque d'être imprévisible d'une exécution à l'autre. De plus, les termes du type *environ*, *beaucoup*, *pas trop* et *à peu près* sont intraduisibles et proscrites au niveau d'un langage informatique.

Une **opération** est **bien définie** si son **résultat** est **entièrement prévisible**. Des versions « bien définies » des exemples ci-dessus pourraient être : *ajouter 2 cl de vinaigre, ajouter 5 g de sel et 1 g de poivre, laisser cuire 65 minutes dans un four chauffé à 220° C, etc.*

Algorithme : Définition finale

Un **algorithme** est un processus (de calcul) non ambigu, déterministe, fini, exprimé en instructions élémentaires exécutables, et si possible efficace, c.-à-d. qui atteint l'objectif pour lequel il a été conçu dans un temps optimal, quelles que soient les valeurs des données.

Propriétés d'un algorithme

De cette définition, les **propriétés d'un algorithme** sont :

- La non-ambiguïté.
- Le déterminisme.
- Les étapes élémentaires explicites et précises.
- La finitude.

Algorithmique : Définition finale

L'**algorithmique** est l'étude formelle des algorithmes. En particulier, elle étudie la **complexité** des algorithmes qui est une mesure théorique de leurs **performances** [1] indépendamment d'un environnement matériel et logiciel particulier. (Nous reviendrons sur cette très importance notion.)

1. Combien de temps un algorithme met-il pour s'effectuer ? Entre une recette de cuisine qui prend une heure et une recette de cuisine qui prend trois heures pour arriver au *même* résultat gustatif et diététique, laquelle doit-on préférer ?

3 Algorithmes informatiques

Des sections précédentes, nous pouvons définir :

Algorithme informatique
Procédure de résolution d'un problème contenant des **opérations bien définies** portant sur des informations, s'exprimant dans une séquence définie **sans ambigüité**, destinée à être traduite dans un langage de programmation.

Langage de description
Comme toute marche à suivre, un algorithme doit s'exprimer dans un certain langage : à priori le langage naturel mais il y a d'autres possibilités : algorigramme, arbre programmatique ou pseudo-code (cf. section suivante : @[Formuler un algorithme]).

Programme, Langage de programmation
Un **programme** est la représentation d'un algorithme dans un langage plus technique compris par un ordinateur (par exemple : Assembleur, C/C++, Java, Python...). Ce type de langage est appelé **langage de programmation**.

Programme correct
Puisque le programme est la représentation d'un algorithme, il faut que celui-ci soit correct pour que le programme le soit. Un **programme correct** résulte donc d'une démarche logique correcte (**algorithme correct**) et de la connaissance de la **syntaxe** (grammaire du langage) d'un langage de programmation.

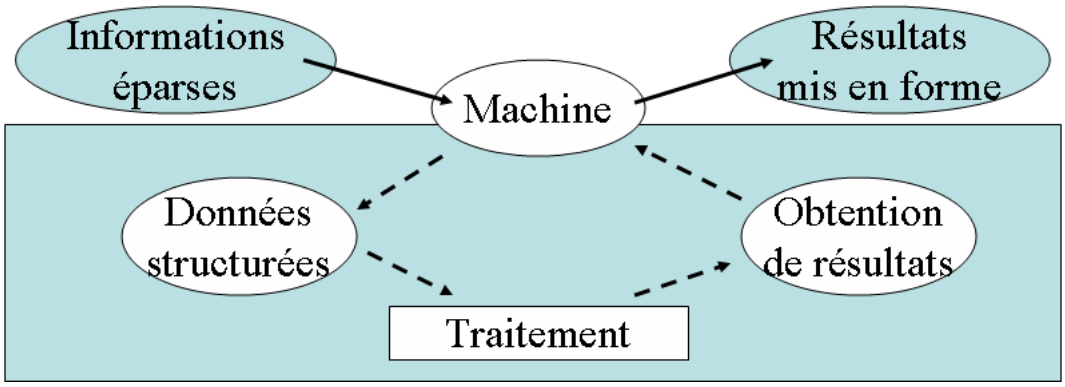
Élaborer des algorithmes corrects
Il est donc indispensable d'**élaborer des algorithmes corrects** avant d'espérer concevoir des programmes corrects.

Fonctionnalités d'une machine
Une **machine**, donc un ordinateur, est dotée de cinq fonctionnalités :

- Entrer les données.
- Exécuter le programme (suite des opérations).
- Séquencer les opérations donc mémoriser le programme.
- Mémoriser les résultats intermédiaires.
- Sortir les résultats.

Propriété
Un algorithme ne dépend **ni du langage** de programmation dans lequel il est implanté, **ni de la machine** qui exécutera le programme correspondant.

De l'importance de l'algorithmique



Comment obtenir un algorithme ?

(Microsoft-Office Clipart)

(Mal)heureusement il n'existe aucune recette.

4 Formuler un algorithme

Lors de la conception d'un algorithme, il est important de se focaliser sur la logique de résolution, sans être perturbé par les détails des langages de programmation.

Sur le problème de la recherche d'un partenaire pour danser, voici deux façons usuelles de les décrire :

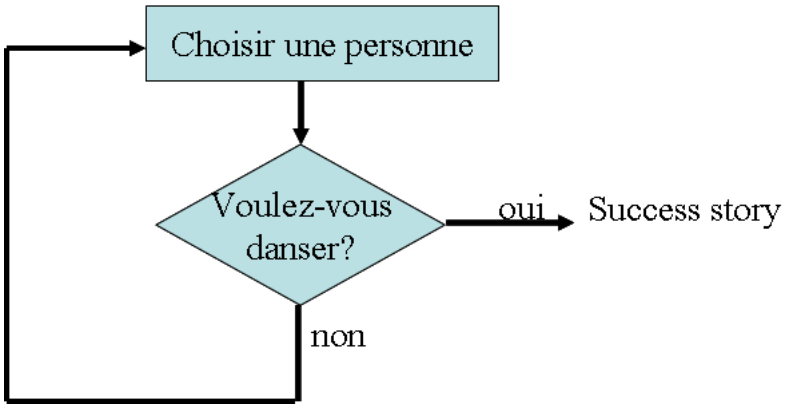
- L'algorigramme
- Le pseudo-code

4.1 L'algorigramme

Algorigramme
Représentation graphique d'un algorithme. Appelé aussi *organigramme de programmation* ou encore *logigramme*.

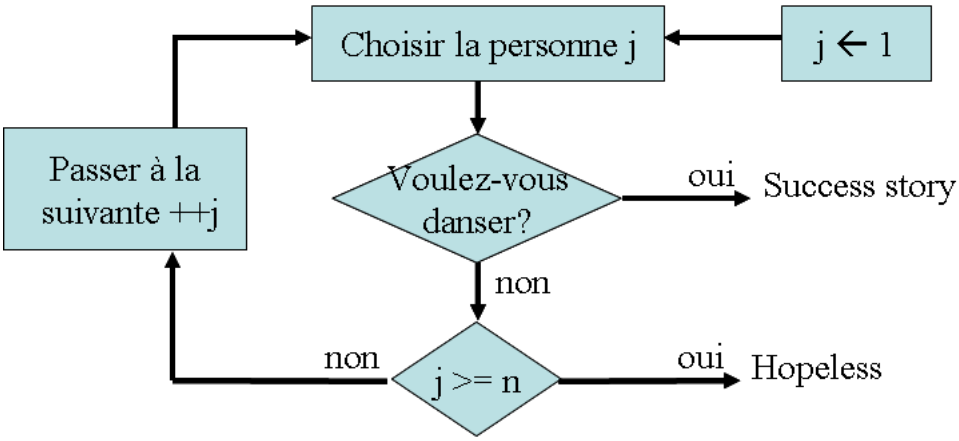
Premier algorigramme : The Dance
(Microsoft-Office Clipart)

Deuxième algorigramme : The Dance
Soyons un peu plus précis. En voici un deuxième mais il n'est pas garanti que l'algorithme puisse se terminer !



Troisième algorigramme : The Dance

En complétant l'algorigramme précédent, celui-ci se termine nécessairement : au pire `n` essais successifs... mais il n'est pas sûr qu'il soit le plus efficace : tant que l'utilisateur n'a trouvé le bon partenaire, il passe au suivant.



Un formalisme qui occupe trop de place

Les traitements (instructions, affectations) sont dans des rectangles, les prises de décision dans des losanges et les flèches permettent d'indiquer l'enchaînement des opérations ou des entrées/sorties. Théoriquement, deux ovales marquent le début et la fin.

4.2 Le pseudo-code

Pseudo-code ou langage algorithmique

Représentation textuelle d'un algorithme. Dit aussi *Langage de Description des Algorithmes* (LDA en abrégé), c'est un langage formel et symbolique qui utilise :

- Des **noms symboliques** destinés à représenter les objets sur lesquels s'effectuent des actions.
- Des **mots-clés** et des **opérateurs** qui traduisent les opérations exécutables par un exécutant donné.

Il est à la base de tous les langages de programmation impératifs.

Pseudo-code : The Dance

```
Algorithme TheDance
Variable j , n : Entier
Variable succes : Booléen
Début
  |  Saisir ( n )
  |  j <- 1
  |  succes <- Faux
  |  TantQue ( j <= n Et Non succes ) Faire
  |    |  choisir la personne j
  |    |  succes <- Voulez - vous danser ?
  |  FinTantQue
  |  Si ( succes ) Alors
  |    |  Afficher ( "Histoire Heureuse" )
  |  Sinon
  |    |  Afficher ( "Sans espoir..." )
  |  FinSi
Fin
```

Conventions d'écriture

Définir un langage algorithmique est toujours délicat puisqu'il faut trouver l'équilibre entre la lisibilité offerte au lecteur et le formalisme nécessaire au programmeur. En effet, un algorithme est destiné à un lecteur humain qui n'est pas un compilateur. Il doit être compris sans ambiguïté, mais il ne doit pas faire preuve d'une rigidité qui pourrait nuire à une compréhension immédiate.

4.3 Conclusion

Algorithme idéal

Nous dirons qu'un algorithme *idéal*, appelé **algorithme général** et exprimé en pseudo-code, devrait se situer à **mi-chemin** entre la démarche globale exprimée dans un **langage naturel** (langue française) ou structuré (algorigramme) et l'algorithme ultime, c.-à-d. le **programme** exprimé en langage de programmation. Point n'est besoin de donner deux descriptions quasi identiques d'un algorithme, l'une dans un langage *pseudo-codé*, l'autre dans un langage de programmation.

Conventions

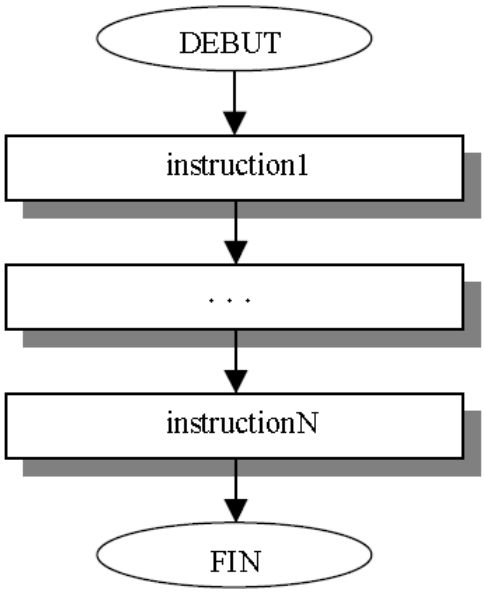
A partir du moment où des conventions sont prises dans un contexte bien déterminé (un service informatique d'une entreprise, un groupe scolaire...), il convient que **tous** respectent ces conventions. Ce sera à chacun de juger jusqu'où il ne faut pas aller trop loin dans la liberté d'écriture.

5 Structures fondamentales d'un algorithme

5.1 La séquence

Première structure fondamentale

C'est la **séquence d'opérations**. On effectue une première étape, puis une deuxième, puis une troisième et ainsi de suite, **une et une seule fois** dans l'ordre où elles apparaissent.



Exemple

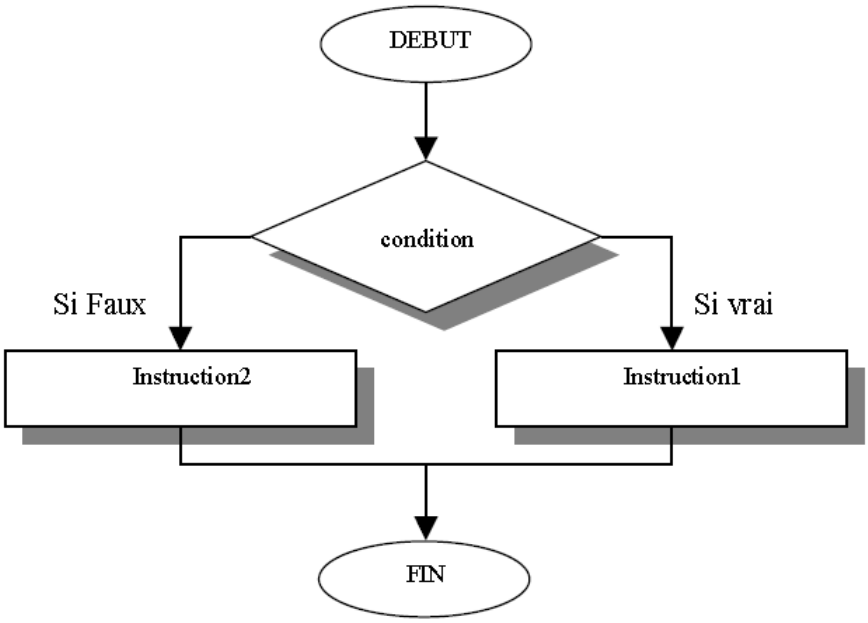
C'est ainsi que pour faire cuire un poisson au court-bouillon :

```
décrocher la casserole
remplir la casserole d'eau froide
plonger le poisson dans la casserole
mettre la casserole sous la plaque
allumer la plaque
cuire le poisson
égoutter le poisson
```

5.2 La structure conditionnelle

Deuxième structure fondamentale

La **structure conditionnelle** permet de choisir si l'exécution aura lieu oui ou non en fonction de la réalisation d'une condition.



Exemple

De cette manière, l'opération « allumer la plaque » (cuisinière électrique ou cuisinière à gaz) pourra être détaillée en :

```
si vous disposez d'une cuisinière à gaz alors
| enflammer une allumette
| tourner le bouton de la cuisinière
| allumer le gaz
sinon # votre cuisinière est électrique
| tourner le bouton de la cuisinière
finsi
```

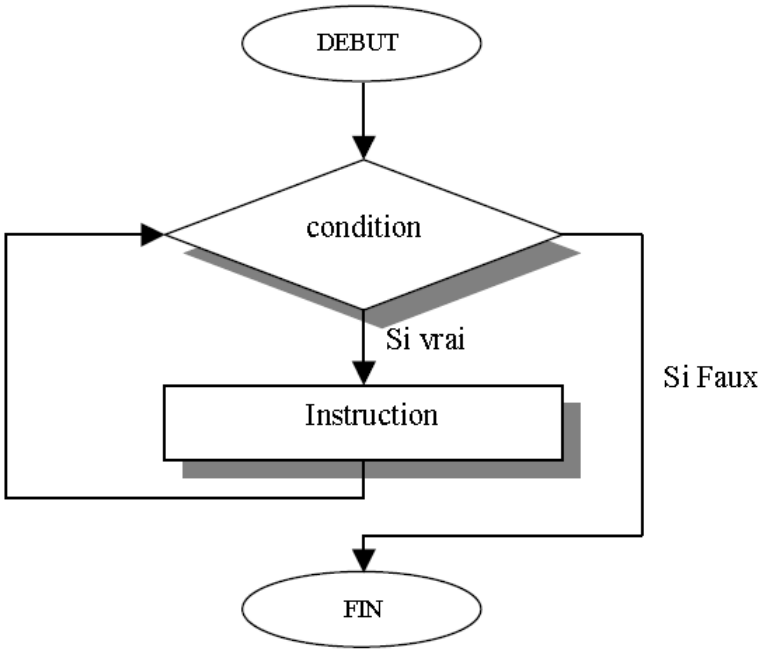
Commentaires

Les phrases après le symbole # ne font pas partie de l'algorithme : elles expliquent au lecteur pourquoi on travaille ainsi. Ce sont des **commentaires**.

5.3 Les répétitives

Troisième structure fondamentale

On distingue les structures **répétitives inconditionnelles** (effectuer une opération un nombre déterminé de fois : tourner le bouton de la plaque de 6 crans vers la droite pour atteindre le thermostat 6 ; attendre 10 fois une minute pour laisser cuire 10 minutes) et les structures **répétitives conditionnelles** (effectuer une opération tant que ou jusqu'à ce qu'une condition soit vérifiée : quand l'eau frémit, tourner le bouton jusqu'au zéro ; faire cuire jusqu'à évaporation complète...).



Exemple

Ainsi, l'opération « cuire le poisson » pourra être détaillée en :

```
attendre que l'eau frémisse
tourner le bouton de la cuisinière de 4 crans vers la gauche
attendre 10 fois une minute
tourner le bouton de la cuisinière vers la gauche jusqu'au zéro
```

Remarque

Dans une répétitive conditionnelle, réside le danger de **boucle infinie** due à une mauvaise formulation de la condition d'arrêt. Par exemple : « Lancer le dé jusqu'à ce que le point obtenu soit 7 » : un humain comprend que la condition est impossible à réaliser, mais un robot appliquant cette directive lancera le dé perpétuellement.

5.4 Thèse de Church-Turing

Thèse de Church-Turing
Elle assure que tout algorithme peut être décrit à l'aide des trois structures :

- La séquence.
- La sélective `Si`.
- La répétitive `TantQue`.

Conséquence
Tout langage de description d'algorithme (et donc tout langage de programmation) doit contenir ces trois structures et peut se réduire à elles. Le pseudo-code décrit et les langages de programmation retenus (C/C++, Java, Python) sont des parfaits exemples de cette thèse.

Exemple
L'algorithme de la cuisson s'écrit finalement :

```
décrocher la casserole
remplir la casserole d'eau froide jusqu'à ce qu'elle soit pleine
plonger le poisson dans la casserole (tant pis si cela déborde)
mettre la casserole sous la plaque
si vous disposez d'une cuisinière à gaz alors
| enflammer une allumette
| tourner le bouton de la cuisinière de 6 crans vers la droite
| allumer le gaz
sinon # votre cuisinière est électrique
| tourner le bouton de la cuisinière de 6 crans vers la droite
finsi
attendre que l'eau frémisse
tourner le bouton de la cuisinière de 4 crans vers la gauche
attendre 10 fois une minute
tourner le bouton de la cuisinière vers la gauche jusqu'au zéro
égoutter le poisson
```

6 Programmation modulaire

6.1 L'analyse descendante

Analyse descendante (*top-down-design*)
Elle consiste en :

> « Décomposer tout problème en (un grand nombre de) petits problèmes élémentaires que toute personne doit être capable de résoudre ».

Ce découpage de **haut en bas** (ou du plus général au plus précis) décompose l'organisation du programme en niveau successifs et réduit la complexité du développement en progressant étape par étape. Cela facilite la compréhension du code et sa testabilité. Elle requiert :

1. Le découpage du problème en sous-problèmes.
2. Pour chaque sous-problème, l'inventaire des données d'entrée et des résultats, puis la spécification du moyen de passer des données d'entrée aux résultats en utilisant les outils de l'algorithmique.
3. La vérification du bon fonctionnement de chaque sous-problème et la validation de ce sous-problème.
4. La mise en place de l'ordre d'exécution des sous-problèmes entre eux pour donner la solution au problème final.
5. La vérification de la solution finale afin de la valider (c.-à-d. l'obtention des bons résultats à partir des données d'entrée).
6. L'optimisation éventuelle de la solution.

En algorithmique
L'analyse descendante est une des techniques fondamentales.
Elle demande :

1. Un **esprit d'analyse** pour décomposer un problème en problèmes plus simples : certains cuisiniers de génie sont incapables d'expliquer une de leurs recettes, de même que certains grands chercheurs ne font pas forcément de bons enseignants.
2. Une **connaissance** des capacités **du public visé** pour savoir jusqu'à quel point il est nécessaire de pousser la subdivision des problèmes : un livre de recettes sera complètement différent selon qu'il s'adresse à des cuisiniers de restaurants, des cuisiniers expérimentés ou des cuisiniers débutants ; de la même façon, selon qu'un algorithme informatique est destiné à être implémenté dans un langage de bas niveau ou dans un langage de très haut niveau, le niveau de raffinement de l'algorithme sera tout à fait différent.

Un autre avantage de la méthode découle de la modularité à laquelle celle-ci conduit naturellement : la cuisson du brochet au court-bouillon pourra s'adapter à la truite au bleue, le beurre blanc s'accommodera à d'autres recettes (beurre blanc au gingembre...).
La **réutilisabilité des modules** est un objectif qui doit être poursuivi le plus souvent possible en algorithmique : « les efforts d'aujourd'hui doivent être encore utiles pour les tâches de demain, les problèmes d'aujourd'hui doivent être simplifiés par les travaux d'hier. »

6.2 Exemple : Analyse descendante

Imaginons que nous voulions expliquer la recette du brochet au beurre blanc.

Haut niveau

Une première approche de notre recette est :

```
faire cuire le poisson
faire un beurre blanc
napper le poisson avec le beurre blanc
```

Cette recette s'adresse à un spécialiste de la cuisine qui sait faire cuire le poisson et faire un beurre blanc.

Cuisinier expérimenté

Si nous nous adressons à un cuisinier expérimenté, nous détaillerons les deux premières opérations :

```
faire cuire le poisson:
| mettre le poisson à l'eau froide
| amener l'eau à frémissement
| baisser le feu
| laisser cuire le poisson 10 minutes
```

```
faire un beurre blanc:
| couper le beurre en morceaux et le mettre au congélateur
| faire une réduction d'échalotes au vin blanc ou au vinaigre
| ajouter à cette réduction les morceaux de beurre bien froids
| faire fondre le beurre dans un bain-marie chaud en remuant rapidement au fouet
```

Cuisinier débutant

Pour un cuisinier débutant, nous serons amenés à détailler encore plus certaines opérations. Par exemple :

```
faire une réduction d'échalotes:
| éplucher et hacher finement 6 échalotes grises
| les mettre dans une casserole
| recouvrir de vin blanc ou de vinaigre
| faire réduire à feu moyen jusqu'à ce qu'il reste environ 3 c. à soupe de liquide
```

Personne ou enfant

Si maintenant, nous nous adressons à une personne ou un enfant qui fait pour la première fois la cuisine, nous détaillerons encore certaines opérations :

- Comment allumer et régler les plaques de cuisson.
- Comment régler la minuterie.
- Comment éplucher les échalotes sans trop pleurer.
- Comment se servir du hachoir sans y laisser un doigt.

Robot ou ordinateur

Si enfin nous nous adressons à un robot ou à un ordinateur, il nous faudra encore détailler toutes les opérations :

- Comment mettre le poisson à l'eau froide.
- Comment baisser le feu.
- Comment remuer au fouet.

Nous décomposerons chaque opération en opérations tellement élémentaires qu'elles seront toutes à sa portée.

7 Un exemple d'algorithme mathématique : le pivot

Pour illustrer la méthodologie d'analyse descendante, nous allons partir d'un problème relativement complexe : la résolution d'un système de Cramer par la méthode d'élimination, ou méthode du pivot de Gauss.

7.1 La méthode du pivot

Rappelons d'abord ce dont il s'agit. Nous partons d'un système de n équations à n inconnues $x_1, \ldots, x_n$ qui peut s'écrire :

$$\begin{cases} a_{11}x_1 & + & \cdots & + & a_{1n}x_n & = b_1 \\ & & \cdots & & & = \cdots \\ a_{n1}x_1 & + & \cdots & + & a_{nn}x_n & = b_n \end{cases} \tag{1}$$

où les nombres a_{ij} sont donnés (en entrée de l'algorithme). Nous allons éliminer successivement les inconnues dans les équations jusqu'à parvenir à un système triangulaire du type :

$$\begin{cases} a_{11}x_1 & + & a_{12}x_1 & + & \cdots & + & a_{1n}x_n & = b_1 \\ & & a_{22}x_1 & + & \cdots & + & a_{2n}x_n & = b_2 \\ & & & & & & \cdots & = \cdots \\ & & & & & & a_{nn}x_n & = b_n \end{cases} \tag{2}$$

qui se résout à partir du bas : la dernière équation permet de calculer x_n que l'on reporte alors dans les équations précédentes et ainsi de suite.

Pour passer du système général (1) au système triangulaire (2) :

- On choisit une équation telle que le coefficient de x_1 (le pivot) soit non nul.
- On porte cette équation en première position.
- Alors en soustrayant à chacune des équations suivantes la première multipliée par un certain facteur (à savoir le quotient du coefficient de x_1 dans l'équation par le pivot choisi) on éliminera x_1 de toutes les équations à partir de la deuxième.
- Il suffit alors de recommencer avec x_2 dans les $n-1$ dernières équations et ainsi de suite pour aboutir au système (2).

Pour que l'algorithme fonctionne, il faut qu'à chaque étape l'on puisse trouver un pivot non nul (c'est le cas si l'on a effectivement un système de Cramer). En fait puisque l'on doit diviser par le pivot, pour minimiser les erreurs de calcul il est préférable de choisir le pivot le plus grand possible en valeur absolue.

7.2 Exemple : Le pivot

Soit le système suivant :

$$\begin{cases} x_1 & +x_2 & +x_3 & = & 1 \\ x_1 & +2x_2 & +4x_3 & = & 0 \\ x_1 & +3x_2 & +8x_3 & = & -1 \end{cases}$$

Dans la colonne de x_1, le plus grand coefficient en valeur absolue est 1 (première équation par exemple). On élimine alors la première inconnue dans la deuxième et la troisième équation :

$$\begin{cases} x_1 & +x_2 & +x_3 & = & 1 \\ & x_2 & +3x_3 & = & -1 \\ & 2x_2 & +7x_3 & = & -2 \end{cases}$$

A partir de la deuxième ligne, dans la colonne de x_2, le plus grand coefficient en valeur absolue est 2 (troisième équation). On permute les équations et on obtient :

$$\begin{cases} x_1 & +x_2 & +x_3 & = & 1 \\ & 2x_2 & +7x_3 & = & -2 \\ & x_2 & +3x_3 & = & -1 \end{cases}$$

On élimine la seconde inconnue de la troisième équation :

$$\begin{cases} x_1 & +x_2 & +x_3 & = 1 \\ & 2x_2 & +7x_3 & = -2 \\ & & x_3 & = 0 \end{cases}$$

qui se résout en $x_3 = 0$ puis $x_2 = -1$ puis $x_1 = 2$.

7.3 Analyse 1

Faisons une première formulation du système :

$$\begin{cases} a_{11}x_1 & + & \cdots & + & a_{1n}x_n & = b_1 \\ & & \cdots & & & = \cdots \\ a_{n1}x_1 & + & \cdots & + & a_{nn}x_n & = b_n \end{cases}$$

en introduisant les matrices :

$$\begin{cases} A & = (a_{ij})_{1 \leq i,j \leq n} \\ X & = (x_i)_{1 \leq i \leq n} \\ B & = (b_i)_{1 \leq i \leq n} \end{cases}$$

D'où l'algorithme :

```
Entrer les données A et B
Trianguliariser le système
si le système est de Cramer alors
| résoudre le système
| sortir le résultat
finsi
```

Nous distinguons quatre grandes étapes dans notre méthode.
Fractionnons les problèmes.

7.4 Analyse 2

Dans la suite nous noterons `a[i,j]` pour a_{ij}, `b[i]` pour b_i et `x[i]` pour x_i.

```
Entrer les données:
| entrer A
| entrer B
```

```
Trianguliariser:
| pour chaque équation faire
| | chercher un pivot maximum
| | amener l'équation du pivot à la bonne position
| | éliminer dans les équations suivantes l'inconnue correspondante
| finpour
```

```
Résoudre:
| pour chaque équation (à partir de la dernière), calculer x[i]
```

```
Sortir les résultats:
| pour chaque i, afficher x[i]
```

Bien que le problème soit déjà bien décomposé, continuons notre analyse.

7.5 Analyse 3

Nous noterons « <- » pour l'expression « mettre dans le terme de gauche la valeur du terme de droite ». C'est ainsi que :

- « `x <- 2` » signifie « mettre dans `x` la valeur 2 »
- « `x <- y` » signifie « mettre dans `x` la valeur de `y` »

Nos quatre étapes se ré-écrivent comme suit :

```
Entrer A:
| pour i <- 1 à n faire
| | pour j <- 1 à n faire
| | | entrer a[i,j]
| | finpour
| finpour
```

```
Entrer B:
| pour i <- 1 à n faire
| | entrer b[i]
| finpour
```

```
Trianguliariser:
| pour i <- 1 à n faire
| | chercher un pivot dans la i-ème colonne (depuis la i-ème ligne)
| | si pas de pivot alors
| | | arrêter
| | sinon
| | | k <- numéro de ligne du pivot
| | | échanger la i-ème ligne et la k-ème ligne
| | finsi
| | éliminer x[i] de la (i+1)-ème ligne à la dernière
| finpour
```

```
Résoudre:
| pour i <- n à 1 pas -1 faire
| | calculer x[i]
| finpour
```

```
Sortir les résultats:
| pour i <- 1 à n faire
| | afficher x[i]
| finpour
```

Certaines de ces opérations deviennent ici suffisamment simples pour être effectuées par l'ordinateur. *Entrer* est une saisie au clavier, *afficher* est un affichage sur l'écran ; dans les deux cas il s'agit de nombres réels, et tout langage de programmation évolué est capable d'effectuer ces instructions. Les autres étapes sont encore trop complexes pour pouvoir être effectuées. Détaillons-les.

7.6 Analyse 4

Pour chercher un pivot maximum, nous parcourons toute la i-ème colonne à partir de la i-ème ligne en retenant : dans « `vmaximum` » la valeur maximum rencontrée jusqu'à un instant donné, et dans « `ligne` » le numéro de la ligne dans laquelle ce candidat au pivot a été rencontré.

```
chercher un pivot dans la i-ème colonne:
| vmaximum <- 0 # au départ le maximum rencontré est 0
| ligne <- i # la i-ème ligne est candidate au pivot
| pour k <- i à n faire
| | si abs(a[i,k]) > vmaximum alors # on a trouvé mieux
| | | ligne <- k # on garde le numéro de la ligne
| | | vmaximum <- abs(a[i,k]) # et le nouveau maximum
| | finsi
| finpour
| si vmaximum = 0 alors
| | pas de Cramer # la méthode ne s'applique pas
| sinon
| | retourner ligne # c'est le numéro de la ligne du pivot
| finsi
```

```
échanger la i-ème ligne et la k-ème ligne:
| pour j <- 1 à n faire
| | stocker a[i,j] # sinon on va perdre sa valeur à la ligne suivante
| | a[i,j] <- a[k,j]
| | a[k,j] <- valeur stockée
| finpour
```

```
éliminer la i-ème variable dans la k-ème équation:
| facteur <- a[k,i] / a[i,i]
| soustraire à la k-ème ligne la i-ème multipliée par facteur
| b[k] <- b[k] - facteur * b[i] # modification du deuxième membre
```

```
calculer x[i]:
| calculer (b[i] - sigma(a[i,j]*x[j], j=i+1..n)) / a[i,i]
| stocker le résultat dans x[i]
```

Il ne reste qu'à détailler les derniers points encore trop complexes.

7.7 Analyse 5

```
soustraire à la k-ème ligne la i-ème multipliée par facteur:
| pour j de 1 à n faire
| | a[k,j] <- a[k,j] - facteur * a[i,j]
| finpour
```

```
calculer (b[i] - sigma(a[i,j]*x[j], j=i+1..n)) / a[i,i]:
| somme <- b[i]
| pour j <- i+1 à n faire
| | somme <- somme - a[i,j] * x[j]
| finpour
| somme <- somme / a[i,i]
```

Notre analyse est achevée puisque nous arrivons à des opérations exécutables (somme et produit de nombres réels ; faire varier un indice dans un intervalle ; exécuter une instruction si une condition est vérifiée et une autre sinon).

8 Conclusion

Citons à nouveau [Cormen-AL1] : « Un bon algorithme est comme un couteau tranchant — il fait exactement ce que l'on attend de lui, avec un minimum d'efforts. L'emploi d'un mauvais algorithme pour résoudre un problème revient à essayer de couper un steak avec un tournevis : vous finirez sans doute par obtenir un résultat digeste, mais vous accomplirez beaucoup plus d'efforts que nécessaire et le résultat aura peu de chances d'être esthétiquement satisfaisant. »

Le but d'un cours d'**algorithmique** est :

1. **Définir une bonne démarche d'élaboration d'algorithmes**.
2. Comprendre l'intérêt d'utiliser certaines méthodes ou **techniques** classiques qui ont fait leurs preuves.

Le tout devrait avoir pour résultat l'élaboration de **bons programmes**, c.-à-d. *des programmes dont il est facile de se persuader qu'ils sont corrects* et des programmes dont la maintenance est la plus aisée possible. Dans ce sens, un tel cours se situe en amont des cours de **langage de programmation**.

Ainsi afin d'envisager la résolution d'une multiplicité de problèmes prenant leur source dans des domaines différents, ce cours envisage l'étude des points suivants :

- La programmation structurée : les variables et les structures de contrôle
- La programmation procédurale : les sous-programmes et le passage de paramètres
- La logique de traitement des tableaux
- La logique de traitement des fichiers séquentiels
- La programmation orientée objet
- La résolution de problèmes récursifs
- La logique de traitement des structures de données particulières telles que listes, files d'attente, piles, tables de hachage, arbres, graphes, etc.

9 Références générales

Ressources du cours

L'exemple [Le pivot] est tiré de [Monasse-KM1]. On trouvera des lectures complémentaires dans [Warin-PG1 :c1]. Les ouvrages de [Cormen-AL1] et surtout ceux de [Knuth-AL] sont les références en algorithmique.

Cori-X1 @MiscCori-X2, author = Robert Cori and Jean-Jacques Lévy, title = Algorithmes et Programmation, howpublished = http://www.enseignement.polytechnique.fr/profs/informatique/Jean-Jacques.Levy/poly/, note = Cours de l'École Polytechnique

Cormen-AL1 @BookCormen-AL1, author = Cormen, T.H. AND Leiserson, C.E. AND Rivest, R.L. AND Stein, C., title = Algorithmique, publisher = Dunod Informatique, ISBN = 2-10-003922-9, year = 2010 (3e édition), type = Algorithmique fondamentale, note = Cours avec 957 exercices et 158 problèmes – Compléments en ligne www.dunod.com

Knuth-AL2 @BookKnuth-AL2, author = Knuth Donald E., title = Seminumerical Algorithms, publisher = Addison-Wesley, Reading, Mass., year = 1981, isbn = , type = , series = The Art of Computer Programming, volume = 2 edition = , school = , note = Ouvrage de référence

Knuth-AL3 @BookKnuth-AL3, author = Knuth Donald E., title = Sorting and searching, publisher = Addison-Wesley, Reading, Mass., year = 1975, isbn = , type = , series = The Art of Computer Programming, volume = 3 edition = , school = , note = Ouvrage de référence

Monasse-KM1 @BookMonasse-KM1, author = Monasse, Denis, title = Option informatique - SupMPSI, publisher = Vuibert, ISBN = 2-7117-8831-8, year = 1996, type = Enseignement supérieur Et Informatique, edition = , school = , note =

Warin-PG1 @BookWarin-PG1, author = Warin, Bruno, title = L'algorithmique - passeport informatique pour la programmation, publisher = Ellipses, year = 2002, 1è Edition, isbn = 2-7298-1140-0, type = , edition = , school = , note =

Ressources complémentaires

Pour prolonger votre réflexion sur le concept d'algorithme (juillet 2017) :

- Les Sépas 18 : Les algorithmes : https://www.youtube.com/watch?v=hG9Jty7P6Es
- Les Sépas 11 : Un bug : https://www.youtube.com/watch?v=deI0GV5sWTY
- Le crépier psycho-rigide : https://pixees.fr/?p=446
- Le baseball multicouleur : https://pixees.fr/?p=450
- Le jeu de Nim : https://pixees.fr/?p=443