Deuxième partie

Le cours

Plan du cours

1. Introduction (2hr)
   - algorithme
   - Programmation : linéaire, récursive, non-linéaire
2. Langage C (6hr)
   - nature du langage
   - types de variables
   - instructions de branchement
   - variables composées
3. Système d'exploitation Unix (0.5hr) (+ TP 1)
4. Langage C (4hr)
   - fonctions, bibliothèques
   - entrées, sorties
   - Pointeurs

Il est prévu d'effectuer 20hrs de CI + 8 hrs de TP (minimum de 4 TP).

CHAPITRE

3

Introduction

3.1 Notions d'algorithmes

Définition pratique :

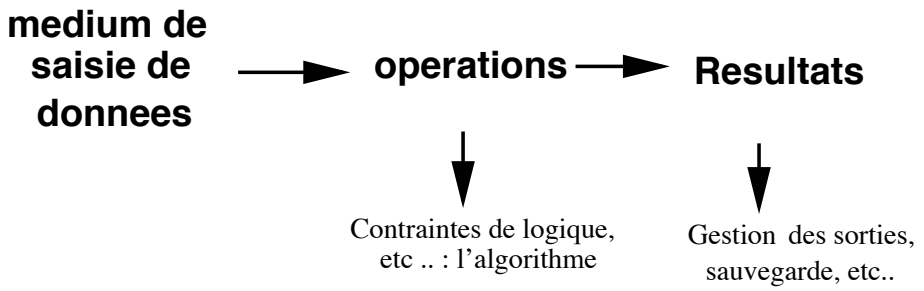
> Un *algorithme* ⇔ langage mathématique : liste d'instructions à effectuer suivant des contraintes de logique. Cette liste doit avoir un *début*, et une *fin*. Le nombre d'instructions effectuées lors de sa mise en œuvre doit être *fini* (dénombrable).

Exemples d'algorithmes : règles de l'arithmétique, grammaire française, etc ..

Mise en œuvre d'un algorithme : il s'agit de codifier (comme un texte de loi) les opérations que nous désirons effectuer pour les rendre parfaitement logiques, sans contradiction. Il est ensuite nécessaire de *programmer* (ou 'coder') notre algorithme (algo) pour le rendre compréhensible à un outil de travail : l'ordinateur.

• *Le défi est d'ébaucher un algo sans faille, sans erreur, car l'ordinateur ne peut corriger ces erreurs par introspection.* Bref, un ordi n'est pas un substitut pour l'intelligence humaine !

Le point de départ est d'identifier les opérations à effectuer pour résoudre un problème donné. Ensuite il faut identifier les variables ou paramètres qui vont nous permettre de résoudre ce problème. Voir le schéma no. 1.



Schema 1

3.2 Programmation

Avant de définir une stratégie pour la programmation, il convient de soumettre le problème en cause à une étude approfondie : la nature du problème guide le programmeur dans l'élaboration d'un algorithme.

3.2.1 Programmation linéaire :

le nombre de solutions ou de résultats est directement proportionnel au nombre d'opérations ou de variables.

Exemple : en économie, élaboration d'un budget de gestion ; en mathématiques, une relation de fonction telle que

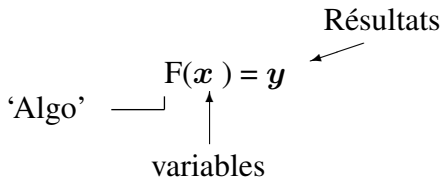
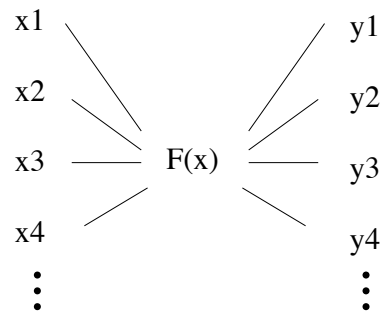


FIG. 3.1 – Illustration du principe de la programmation linéaire.

Ici la fonction 'F' symbolise l'ensemble des opérations à effectuer sur les variables ou paramètres dénotés $\boldsymbol{x}$. Le ou les résultats sont assigné(s) à $\boldsymbol{y}$. Voir la Fig. 3.1 et le schéma no. 2.

Exemples simples : les expressions

$$\boldsymbol{y} = 2\boldsymbol{x} + 1$$



Schema 2

et

$$\boldsymbol{y} = \sin(\boldsymbol{x}).$$

Dans le premier exemple : un paramètre (=1), deux opérations (addition et multiplication), une variable (x), un résultat (y). L'égalité ⇒ opération d'affectation du résultat des opérations à la variable y. L'ensemble de toutes les opérations constitue un algorithme.

Un algorithme linéaire permet deux types de stratégie pour la programmation :

- *calcul sériel* → à la chaîne, une opération à la fois ;
- *calcul parallèle* → plusieurs opérations faites simmultanément en temps réel.

Se référer au schéma no. 3 vu en cours.

Avantages du calcul sériel :

- simplicité de programmation
- portabilité
- un seul ordinateur : économique ..
- pas de gestion des communications : biblio, réseau ..

Avantages du calcul parallèle :

- réduction du temps de calcul (efficace)
- utilisation rationnelle des ressources : e.g. Grille
- ordinateurs spécialisés : multiprocesseurs Cray, BEOwolf

Le choix de la stratégie de programmation peut dépendre du type de variables manipulées, de la nature du problème.

Par exemple : évaluons la norme d'un vecteur de nombres par son produit scalaire.

$$\boldsymbol{X} \rightarrow n \ \ entrees$$

$$y = F(\boldsymbol{X}) = \boldsymbol{X} \cdot \boldsymbol{X} = \sum_{i}^{n} x_i^2 = x_1^2 + x_2^2 + ...x_n^2$$

On reconnaît la même opération de multiplication répétée n fois :

$$y = x_1 \cdot x_1 + x_2 \cdot x_2 + ...$$

Ce type d'opérations se prête donc parfaitement au calcul parallèle. Cf. le schéma 3 vu en cours.

Question : est-ce que la distribution de n opérations de multiplication vers n ordinateurs est toujours optimale ?

La réponse dépend de plusieurs paramètres : n, vitesse d'exécution des ordinateurs, communications réseau, espace mémoire vive de l'ordinateur, etc ...

3.2.2 Récursivité :

On reconnaît un algorithme récursif à une suite d'opérations dont le résultat fait intervenir la suite elle-même ; lorsqu'il s'agit d'une fonction, celle-ci s'appelle *elle-même* au cours de l'exécution.

Mathématiquement :

$$y = F(\ F(\dots\ F(\boldsymbol{x}))$$

où les points de suspension représente un nombre d'appels à F (soit F répétée un nombre arbitraire de fois).

Exemple : l'associativité des opérations mathématiques. La multiplication

$$y = x^4 = x \cdot x\ \cdot\ x \cdot x$$

peut aussi s'écrire

$$y = x^2 \cdot x^2$$

Appelons $F(x) = x \cdot x$. Alors :

$$y(x) = x^4 = F(x) \cdot F(x) = F(F(x)).$$

On peut développer des algorithmes puissants autour de ces idées : ici la première suite de 4 multiplications demande 4 opérations de multiplications + trois de stockage, pour les résultats intermédiaires.

Par contre, si on applique l'algorithme récursif : $F(x) = x \cdot x$, il faudra 2 multiplications + 2 stockages : l'algo est deux fois plus performant. À revoir en TD.

$x \cdot x \ \cdot \ x \cdot x$
$\underline{\qquad} \cdot x$
Résultat intermédiaire = x^2
$\underline{\qquad\qquad} \cdot x$
Résultat intermédiaire = x^3
$\underline{\qquad\qquad\qquad} \cdot x = y$

3.2.3 Programmation non-linéaire

On rencontre parfois des problèmes où il est impossible d'exprimer la solution recherchée en termes de variables indépendentes ou paramètres. Ceci suggère une définition pratique pour cette classe de problèmes :

La solution dépend d'elle-même (grosso modo)

Problèmes types : les équations aux valeurs propres. Mathématiquement :

$$y = \lambda^{-1} F(\boldsymbol{x}, y)$$

Note : si $F(\boldsymbol{x}, y)$ est une dépendance linéaire de y, alors le problème se ramène à ce que nous avons vu en (1).

Toute solution y satisfaisant notre équation ci-haut revient à rechercher les zéros de la fonction G définis par

$$G(\boldsymbol{x}, y) \equiv F(\boldsymbol{x}, y) - y = 0$$

Type de problème plus difficile à traiter car il faut démontrer l'unicité de la solution y pour un vecteur de paramètres $\boldsymbol{x}$ donné.

Exemple : l'équation quadratique (second degré en x)

$$y(x) = a\, x^2 + b\, x + c$$

peut être vue de deux façons :

(a) Pour tout x, une seule solution y : d'où relation de linéarité
(b) Pour tout y, deux solutions x : non-linéarité (c'est le théorême de Gauss)

Le polynôme $y(x)$ peut toujours s'écrire

$$G(\boldsymbol{x}, y) = y - a\, x^2 - b\, x - c = a\, x^2 + b\, x + c' = 0$$

pour lequel il existe une solution analytique : ainsi en posant $y = 0$ on retrouve la solution bien connue pour x

$$x = -\frac{b}{2a} \pm \frac{1}{2a}\sqrt{b^2 - 4ac'}$$

avec $c' = c - y$. En général il n'existe pas d'expression analytiques pour des puissances élévées, et il faut développer un algorithme numérique pour la recherche des racines.

Autre exemple : l'équation $y = \sin(x\,y)$. Elle peut s'écrire $G(x, y) = \sin(xy) - y = 0$.

(a) Pour x constant (fixe) : $0 < x < 1$, une seule solution, $y = 0$
En effet pour $y = 0$

$$\frac{\mathrm{d}\sin(xy)}{\mathrm{d}y} = \cos(xy)\; x = x\;.$$

et

$$\frac{\mathrm{d}y}{\mathrm{d}y} = 1$$

pour toute valeur de y. Le nombre de solutions augmentera avec $x > 1$. Voir schéma 4 fait en cours.

(b) Si par contre y est fixe $\Rightarrow$ recherche x comme variable libre.
• Aucune solution possible si $|y| > 1$. Inutile de rechercher une solution.
• Si $|y| < 1$, alors on peut espérer une solution au moins. Appelons là $x = x_s$. Mais puisque $\sin(x_s y) = \sin(x_s y + 2n\pi)$ pour tout entier n, alors toute valeur de x telle que $x = x_s + (2n\pi)/y$ sera également solution. Il y a maintenant une infinité de solutions à trouver.

Quelques remarques s'imposent de notre recherche de solutions :
– étudier la nature du problème pour restreindre le domaine de notre recherche, notre algorithme.
– inutile de développer un algo si aucune ou une infinité de solutions existent.
– identifier toutes instructions (ou opérations) à effectuer, incluant la saisie de paramètres.