

Algorithme et programme

Introduction

Objectif : résolution de problèmes sur l'ordinateur via un traitement automatique de l'information

Ordinateur : c'est une simple machine qui a des contraintes matérielles et des limitations (capacité, mémoire, calcul...). De plus, celle-ci ne fait que ce que l'on dit de faire.

Programme : c'est un moyen de communication entre une machine et une personne (le programmeur)

Problème à résoudre =====> algorithme =====> programme
(analyse) (implémentation)

Important : un algorithme est indépendant du langage.

Différentes phases de développement

1. Problème à résoudre en langage naturel (spécification, cahier des charges)
2. L'analyse du problème permet de passer à l'algorithme.
3. Programmer, c'est traduire l'algorithme dans un langage compréhensible par l'ordinateur (C, Java, Maple, Pascal par exemple)

données ==> algorithme ==> résultats
(analyse)

Analyse : c'est une suite d'étapes qui conduisent à l'algorithme.

La première étape primordiale est de définir les informations à traiter, les données (glossaire).

L'analyse descendante consiste à décomposer le problème en sous-problèmes plus faciles à résoudre ... jusqu'à obtenir des instructions de base. "*diviser pour régner*"

4. La dernière étape est la **validation** de l'algorithme : les jeux de test.

Exemple : trier une liste d'élèves.

Ordinateur

- Organes d'entrées : clavier, mulot, supports magnétiques ou optiques, tablette graphique
- Organes de sortie : écran, imprimantes, supports magnétiques ou optiques
- unité centrale
 - processeur : rangement des données et du programme
 - mémoire centrale : exécute les instructions
- Supports de stockage
 - supports magnétiques : disque dur, disquette (normale, cochonnerie de ZIP...),
 - supports optiques : CD, DVD
 - supports électroniques : clé USB

Définition d'un algorithme :

Un algorithme, c'est

- est une suite ordonnée d'instructions
- est séquentiel (1 instruction à la fois)
- doit s'arrêter (nombre fini d'opérations)
- déterministe (réponse non aléatoire) : à entrées équivalentes, résultats identiques.

Remarque : il n'y a pas forcément qu'un seul algorithme pour un problème donné. En général, on peut les classer suivant différents critères (performance, occupation mémoire, lisibilité, extensibilité ...)

Langage algorithmique

- Un même langage commun à toute analyse
- Algorithme de principe en langage naturel
- Algorithme
 - langage algorithmique : plus précis, plus simple que le langage naturel
 - organigramme (représentation graphique)

Le langage algorithmique est souvent plus lisible que le langage de programmation (en tout cas, il est indépendant de ceux-ci)

Exemple : calcul du minimum entre deux nombres.

1. Formulation de l'algorithme

a) Affectation (mémorisation)

La mémoire centrale peut être vue comme un ensemble d'emplacements, de cases, de mots. Une case mémoire qui dispose d'un nom permettant de l'identifier est appelée une variable. Une variable est un contenant. Sa valeur le contenu.

L'affectation est une opération élémentaire qui permet de ranger une valeur dans une variable. On la note `:=` ou `<-`.

Exemple : `X := 4`

La variable X conservera la valeur 4 jusqu'à ce qu'une nouvelle instruction lui affecte une nouvelle valeur.

Le terme de gauche d'une affectation est toujours un nom de variable. Le terme de droite peut être une constante (valeur immédiate), une autre variable ou une expression.

On peut très bien envisager différents types de variables : entières, réelles, booléennes ... (type)

EXERCICE 1 : échange de deux variables

EXERCICE 2 : permutation circulaire de trois variables.

La définition d'une variable permet d'écrire des choses comme `i := i + 1`.

b) Lecture/Ecriture

(introduction/restitution, mémorisation)

`lire(A);` signifie qu'il faut mettre dans la case(variable) A la valeur présente sur un périphérique d'entrée de la machine à préciser => saisie des données.

`écrire(A);` signifie placer sur un organe de sortie de la machine à préciser le contenu de la variable A.

La machine lit sur l'organe d'entrée et écrit sur un organe de sortie. L'utilisateur fait l'inverse.

c) L'algorithme conditionnel

Orientation du traitement en fonction des données.

```
SI condition ALORS traitement FSI;  
SI condition ALORS traitement 1  
    SINON traitement 2  
FSI;
```

La condition peut être

- une expression logique : `=`, différent, `<`, inférieur ou égal, `>`, supérieur ou égal, ET, OU , NON, ...
- une variable booléenne : vrai ou faux.

EXERCICE : écrire l'algo qui donne la plus grande valeur de deux valeurs saisies au clavier. Si la méthode choisie fait intervenir plusieurs commandes "afficher", donner une version qui ne laisse qu'une occurrence.

EXERCICE : écrire l'algo qui affiche à l'écran la comparaison de deux variables A et B avec les messages suivants : A < B, B < A ou A=B

NOTE 1 : expliquer différence SI SINON SI et plusieurs SI à la suite.

NOTE 2 : c'est quoi le sinon de SI a=0 ET b=0

Appartenance d'un nombre à un intervalle

SELON expression =

```
valeur 1 : action 1;  
valeur 2 : action 2;  
valeur 3 : action 3;  
défaut    : action 4;
```

FSELON;

EXERCICE : comparer 3 nombres et renvoyer le plus grand A, B, C ---> R

```
LIRE (A, B, C);  
SI (A>=B) ALORS R=A SINON R=B FSI;  
SI (R<=C) ALORS R=C FSI;
```

EXERCICE : solution de l'équation $A X^2 + B X + C = 0$

```
SI (A=0) ALORS  
  SI (B=0) ALORS  
    SI (C=0) ALORS ECRIRE (IR est solution);  
    SINON ECRIRE (pas de sol)  
    FSI;  
  SINON  
    ECRIRE ( 1 solution = -C/B);  
  FSI;  
SINON  
  D = B*B - 4 A*C;  
  SI (D>=0) ALORS  
    SI (D=0) ALORS ECRIRE ( 1 solution = -B/(2A) );  
    SINON ECRIRE ( 2 solutions ...);  
    FSI;  
  SINON ECRIRE (solutions complexes);  
  FSI;  
FSI;
```

d) L'itération

Répétition d'un traitement sur un ou plusieurs objets de même nature. L'arrêt du traitement est réalisé par une condition.

Insistons bien sur les différentes phases : initialisation, test, incrémentation, boucle

i) Algorithme à récurrence fixe : POUR

Pour un algorithme à récurrence fixe, le nombre de répétitions est connu d'avance.

EXEMPLE : calcul de $Y = \sum(A_i)$ pour i de 1 à N , N fixe

LEXIQUE : N, Y, I, A

```
LIRE (N);
Y := 0;
POUR I:=1 JUSQU'A N PAS 1 FAIRE
    LIRE (A);
    Y := Y + A;
FAIT/FINPOUR;

ECRIRE (Y);
```

EXERCICE : calcul de $n!$

ii) Algorithme à récurrence variable : TANT QUE

Le nombre de récurrences non fixé à l'avance est déterminé dans le processus de traitement.

Important : deux formes : **TANT QUE** et **REPETER**

```
TANT QUE condition
    traitement
FAIT/FTANTQUE;
```

```
REPETER
    traitement
JUSQU'A condition FAIT;
```

ATTENTION : il faut être sûr que l'algo se termine **quelles que soient** les entrées.

EXERCICE : saisie de deux nombres positif au clavier.

EXERCICE : écrire le POUR avec un TANT QUE

EXERCICE : division de l'entier a positif par l'entier b strictement positif en supposant que l'on ne connaît que l'addition et la soustraction, i.e. trouver q et r positifs tels que $a = b q + r$ avec $r < b$:

LEXIQUE : A, B, Q, R

```

LIRE (A,B) ;
Q := 0 ;
R := A ;
TANT QUE (R>=B) FAIRE
    Q := Q + 1 ;
    R := R - B ;
FAIT ;
ECRIRE (Q,R) ;

```

EXERCICE : Résolution de l'équation $x=f(x)$ par approximations successives

Partant de x_0 quelconque, on calcule successivement x_1, \dots, x_n jusqu'à ce que $|x_i - x_{i-1}| < \text{EPSILON}$. La convergence est vérifiée si la norme infinie de la dérivée est strictement plus petite que 1.

LEXIQUE : X0, X1, E, MAX, I

```

LIRE (X1,E,MAX) ;
I := 0 ;
REPETER
    X0 := X1 ;
    X1 := f(X0) ;
JUSQU'A ((abs(X1-X0)<E) OU (I>MAX)) FAIT ;
ECRIRE (X0) ;

```

Attention ici à l'indice de boucle. Il se justifie par le fait qu'il peut être difficile de vérifier que la fonction f a la bonne propriété ou alors que l'algorithme "diverge" avec les erreurs dues à la précision machine.

2. Algorithmes paramétrés

Un algorithme paramétré se comporte comme une nouvelle instruction qui sera exécutée avec les valeurs de paramètres d'entrée et qui fournira des valeurs de paramètre de sorties. Deux formes possibles :

- la procédure
- la fonction

a) Procédure

Une **procédure** est un algorithme paramétré qui, à partir des paramètres d'entrée, fournit un ou plusieurs résultats en sortie.

```
procédure Id ( Id1, Id2, ... Idn : E ; Idn+1, Idn+2, ... Idm : S ;  
Idm+1, Idm+2, ... Idq : E/S ) :  
    instruction 1;  
    ...  
    instruction n;  
FIN;
```

Id est l'identificateur de la procédure.

Id₁ à Id_q sont les identificateurs des paramètres formels : on distingue les paramètres en entrée (:E), en sortie (:S), en entrée/sortie (:E/S).

Pour appeler la procédure, on fera par exemple Id (P₁, ..., P_q) ;

b) Fonction

Une **fonction** est un algorithme paramétré qui fournit un résultat unique et qui est utilisé dans une expression.

```
fonction Id ( Id1, Id2, ... Idn ) :  
    instruction 1;  
    ...  
    instruction n;  
  
retourner E;  
FIN;
```

E est une expression qui donne sa valeur à l'instruction.

Pour appeler la fonction, on fera par exemple Résultat := Id (P₁, ..., P_n) ;

EXERCICE: Ecrire sous forme de procédure et de fonction, le minimum de deux valeurs.

EXERCICE: Ecrire l'échange de deux nombres.

3. Les tableaux : un type de structuration des données en mémoire.

Un tableau est une variable structurée qui regroupe sous un même nom plusieurs données "élémentaires".

Notion de structure de données : un ensemble de "cellules" mémoire reliées d'une certaine façon pouvant contenir une valeur (souvent de même type). Au point de vue mémoire, chaque cellule se concrétisera par une ou plusieurs cases mémoire.

a) Tableau de dimension 1

C'est une structure de données formées de cellules contiguës et d'accès direct (i.e. qu'il n'est pas nécessaire de connaître les cellules précédentes pour connaître la valeur d'une cellule). C'est une suite de cellules.

Pour définir un tableau, il faut :

- un nom
- le type de données à indiquer
- le type d'indice
- le nombre d'éléments

On accède à l'élément i du tableau A par $A[i]$ ou $A(i)$.

Dans les programmes, un tableau est souvent surdimensionné. On choisit pour le tableau une taille suffisante pour contenir les éléments voulus. Une variable supplémentaire est souvent utilisée pour donner la taille réelle/effective du tableau.

EXERCICE : trouver le max d'un tableau d'entier.

EXERCICE : somme des éléments d'un tableau

EXERCICE : addition de 2 tableaux

EXERCICE : Vérifier qu'un tableau est trié

EXERCICE : Rotation droite

EXERCICE : insérer un nouvel élément dans un tableau

EXERCICE : Séparer au sein d'un même tableau les valeurs positives des valeurs négatives.

```
procédure SEPAR ( T : E/S, MAX : E ) :
    n := 1;
    p := MAX;
    sortie = FAUX;
```

REPETER

```

TANT QUE (T[n]<0) ET (n<=MAX) FAIRE n:=n+1; FAIT;
// on peut se contenter de tester n<=p
TANT QUE (T[p]>=0) ET (p>0) FAIRE p:=p-1; FAIT;
SI (n<p)
ALORS ECHANGE(T[n],T[p]);
SINON sortie = VRAI;
FSI;
JUSQU'A (SORTIE) FAIT;
FIN;

```

Lexique :

T tableau dont on veut séparer les éléments, indicé de 1 à MAX

MAX nombre d'éléments de ce tableau

n indice du premier élément positif s'il existe

p indice du dernier élément négatif s'il existe

sortie : variable booléenne qui vaut VRAI s'il faut arrêter

b) Tableau à deux dimensions

Le tableau à deux dimensions ou matrice est une structure de données permettant l'implantation d'une suite A doublement indicée par (i,j)

Accès à un élément **A[i, j]** ou **A(i, j)**

Il est de même pour les tableaux à n entrées :

A[i₁,i₂,...,i_n] ou **A(i₁,i₂,...,i_n)**

EXERCICE : les calculs sur les matrices ...