Qu'est-ce qu'un algorithme ?
Comment exprimer et communiquer un algorithme ?
« Opération élémentaire » et la représentation des nombres

Louis Nolin

Avant-Propos. – Dans cette seconde partie de *l'introduction à la programmation des problèmes scientifiques,* après le contexte exposé dans le numéro précédent du bulletin, arrivent les parties :

- Qu'est-ce qu'un algorithme ?
- Comment exprimer et communiquer un algorithme ?
- La notion d'« opération élémentaire » et la représentation des nombres.

Dans un premier temps, la notion d'algorithme n'est pas définie de manière rigoureuse, c'est « adapter une méthode à la résolution automatique d'un problème ».

Ensuite, la manière explicite d'exposer l'algorithme, qui fournit la solution à un problème, se traite avec un schéma, nommé organigramme. Vous pourriez vous orienter vers l'utilisation de LARP pour agrémenter les exercices proposés, le logiciel dispose d'un système d'aide en ligne offrant une description détaillée de l'environnement de développement de LARP ainsi que de son langage pseudo-code et ses instructions d'organigramme. L'environnement de développement de LARP est constitué de plusieurs composants, dont les principaux sont la fenêtre principale, la console d'exécution, la fenêtre d'exécution pas-à-pas.
LARP est en fait un acronyme. Il vient de la compression de la phrase « Logiciel d'Algorithmes et de Résolution de Problèmes ». LARP est un langage de programmation permettant le prototypage rapide d'algorithmes. L'avantage de LARP est que le programme est un langage pseudo-code à syntaxe flexible et non un code source à compiler, ce qui permet de formuler des algorithmes en un langage semi-naturel plutôt que de devoir adhérer à une syntaxe rigide et cryptique telle que celle des langages de programmation traditionnels (C++, Pascal, Java, etc.). (lu sur https://www.tice-education.fr)

La notion d'« opération élémentaire » et la représentation des nombres sont exposées à l'aide des machines de Turing.

Et rappelons, qu'un échauffement à la main du calcul constitue un bon début à l'aide des exercices proposés. (N.D.L.R.)

II - QU'EST-CE QU'UN ALGORITHME ?

On peut définir la programmation comme l'art de rechercher des méthodes de calcul et de les adapter à la résolution automatique des problèmes.

Expliquons-nous.

"Résoudre automatiquement un problème" c'est, finalement, le soumettre à un calculateur automatique particulier, capable d'effectuer quelques opérations simples;

"Adapter une méthode à la résolution automatique d'un problème" c'est donc, essentiellement, la réduire à une chaîne d'opérations élémentaires.

La chose nous est familière.

Dès la classe de 5ème, tous les écoliers s'initient aux méthodes des constructions géométriques, et apprennent par coeur que pour construire la bissectrice d'un angle avec la règle et le compas, il faut exécuter les opérations suivantes :

Tracer un cercle de rayon R , quelconque, dont le centre est le sommet S de l'angle; ce cercle coupe les côtés de l'angle aux points A et B ;

Tracer un cercle de centre A et de rayon R', plus grand que R ;

Tracer un cercle de centre B et de rayon R' ; les deux derniers cercles se coupent au moins en un point C ;

Tracer la droite SC : c'est la bissectrice de l'angle.

La"construction" est ainsi décomposée en une suite d'"opérations permises" :

Tracer un cercle de centre et de rayon choisis;

Tracer une droite passant par deux points choisis.

C'est en 5ème qu'on revoit de façon sérieuse les procédés usuels pour le calcul, mental ou "à la main" , de la somme, la différence, le produit, le quotient à une unité près, de deux nombres exprimés dans la notation décimale. C'est à ces quatre opérations qu'on va ramener par la suite, directement ou indirectement, tous les procédés de calcul comme ceux qui permettent d'extraire la racine carrée d'un nombre positif ou de trouver le p.g.c.d. de deux entiers.

Ce dernier est connu sous le nom d'"algorithme" d'Euclide. Retenons ce terme d'algorithme : il nous servira désormais à désigner ce que nous avons appelé jusqu'ici du nom de procédés, de méthodes de calcul, voire de "formules", comme celles qui donnent la solution d'une équation algébrique de degré inférieur à 5 , la "formule de Simpson" pour le calcul des intégrales définies, ou bien encore les "développements limités" qui permettent d'obtenir des valeurs approchées d'une fonction.

C'est encore un algorithme que l'ensemble des procédés de dérivation formelle. Il comporte des "formules" pour la dérivation des fonctions simples et des "règles" qui permettent d'y ramener, de proche en proche, le calcul des dérivées d'une somme, d'un produit, d'un quotient, d'une composition de fonctions, d'une intégrale, etc.

Les algorithmes auxquels nous venons de faire allusion sont bien connus : ils sont exposés avec plus ou moins de bonheur dans de nombreux ouvrages classiques.

Il n'en est pas toujours de même. Il faut y regarder de plus près pour apercevoir, derrière l'algorithme d'Euclide, celui qui permet d'obtenir la solution des équations diophantiennes de la forme $ax + by = c$; il faut un tout petit peu d'imagination pour adapter les méthodes de STURM à la recherche des éléments propres d'une matrice symétrique tridiagonale; enfin, ce n'est qu'en rapprochant les théorèmes de FROBENIUS et de MISES qu'on peut songer à appliquer le procédé de la multiplication itérée d'une matrice par un vecteur à la recherche des racines des équations algébriques.

Dans le même ordre d'idées on peut constater que, bien souvent, les démonstrations d'existence - pour autant qu'elles sont "effectives" - contiennent implicitement un procédé de construction de l'objet défini : la "méthode des rectangles" pour le calcul approché d'une intégrale est contenue, pour l'essentiel, dans la définition de l'intégrale de Riemann.

Il y a mieux encore. Pendant longtemps, beaucoup de problèmes ont paru si simples qu'on aurait pensé perdre son temps en s'avisant de décrire par le menu un algorithme qui permette de les résoudre. A quoi bon insister, par exemple, sur la manière de ranger une suite de nombres dans l'ordre des valeurs croissantes ? "cela va de soi", n'est-ce-pas ? Oui, sans doute, si le problème est proposé à une personne avertie. Mais pour permettre à un cancre ou à un calculateur automatique de le résoudre, il faudra se donner la peine de décrire soigneusement un algorithme approprié. On constatera alors qu'il y a bien des manières de faire, qui sont loin d'être équivalentes quant à la rapidité d'exécution.

Dans tous les cas, le fait que certaines opérations soient permises à l'exclusion de toutes autres constitue une contrainte qui ne facilite pas les choses, nous le savons par expérience; GAUSS lui-même, à ce qu'on dit, n'était pas peu fier de sa construction de l'angle $2\pi/17$. Et que dire de la "géométrie du compas" où la seule opération permise est la construction de cercles ! (1)

On remarquera que ces démarches présentent de grandes analogies avec la démonstration d'une proposition mathématique.

Démontrer, par les moyens de la géométrie élémentaire, que la projection d'un cercle sur un plan est une ellipse;

Démontrer, par récurrence, l'identité

$$\sum_{m=0}^{n} \cos(mx) = \frac{\sin(\frac{n+1}{2}x)\cos(\frac{n}{2}x)}{\sin\frac{x}{2}} ;$$

Démontrer les axiomes de HILBERT à partir de ceux de VEBLEN ;

---

(1) Voir l'exercice II.2

voilà qui ressemble étrangement aux problèmes que nous avons évoqués.

Nous ne chercherons pas, pour l'instant, à définir de manière rigoureuse la notion d'algorithme : les exemples précédents suffisent pour que nous nous entendions sur ce terme. Nous allons évoquer plutôt les problèmes pratiques posés par la recherche d'algorithmes appropriés à la résolution automatique des problèmes numériques.

EXERCICE II.1

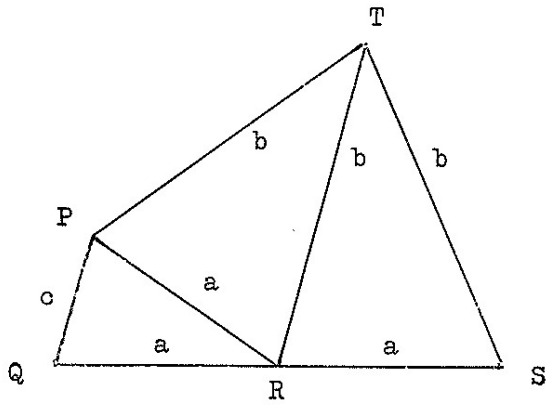
Revoir les algorithmes classiques auxquels on vient de faire allusion.

EXERCICE II.2

Problème de MASCHERONI (ou de NAPOLEON).

Construire le centre du cercle circonscrit à un triangle isocèle en utilisant le compas seul.

On s'appuiera, pour cela, sur les propriétés de la figure suivante :



1. Si on connaît P,R,S ou R,S,T , on peut construire Q ;
2. $bc = a^2$.

III - COMMENT EXPRIMER ET COMMUNIQUER UN ALGORITHME ?

Chacun de nous a pu constater que la nécessité d'effectuer, par exemple, l'addition de deux nombres écrits sur une feuille de papier, déclenche aussitôt une suite de gestes aussi efficaces qu'automatiques. Chez un calculateur chevronné, de nombreux procédés de calcul sont ainsi devenus des mécanismes réflexes.

Mais les algorithmes sont trop nombreux, trop complexes aussi, et parfois trop rarement utilisés pour qu'il en soit toujours ainsi: le problème de leur communication se pose tôt ou tard. Et cela ne va pas sans mal. Qu'on nous demande tout à trac d'enseigner à un jeune élève la manière dont nous calculons, par exemple, la racine carrée d'un nombre entier et nous serons sans doute bien en peine de trouver sur l'heure une expression à la fois correcte et compréhensible pour un algorithme que, par ailleurs, nous connaissons fort bien.

Remplacez le jeune élève par une machine à calculer et vous vous trouverez d'emblée devant l'un des problèmes qui se posent au programmeur : communiquer à une machine, interlocuteur le plus borné qui soit, la suite des démarches qui constitue un algorithme.

C'est qu'en effet le problème de la communication a pour paramètre essentiel la qualité de l'auditeur.

La manière la plus expéditive et la moins fatigante d'exprimer un algorithme consiste à y faire allusion. J'en ai usé largement jusqu'ici; c'est que j'avais choisi des exemples simples et que je m'adressais à des lecteurs avertis.

Malheureusement les algorithmes sont en général compliqués et fort peu connus : il faut, une fois au moins, les exposer de façon détaillée à qui sait "lire, écrire et compter" suivant une expression fameuse.

Méfions-nous donc de ces modes d'expression à peine plus détaillés que la simple allusion. Ainsi, il ne suffit pas de dire que la manière de calculer la valeur de l'expression

$$(1) \qquad a_0x^4 + a_1 x^3 + a_2 x^2 + a_3 x + a_4$$

pour des valeurs données de $a_0,\ldots,a_4$, x est indiquée par l'expression (1) elle-même. Tout algorithme est en effet une suite d'opérations effectuées dans un ordre déterminé. En laissant entendre que l'expression (1) contient tout ce qu'il est possible d'exprimer sur cet enchaînement, voulons-nous dire que nous la considérons comme l'abréviation de l'expression suivante :

$$(2) \qquad \Big[\Big[\Big[\big[a_0(((x.x)x)x)\big] + \big[a_1((x.x)x)\big]\Big] + \big[a_2(x.x)\big]\Big] + \big[a_3.x\big]\Big] + a_4$$

où l'ordre des opérations à effectuer est clairement indiqué ? Non, car nous considérerons comme bien stupide quiconque procédera ainsi; c'est à l'expression

$$(3) \qquad ((((a_0.x) + a_1)\, x + a_2)\, x + a_3)\, x + a_4$$

que nous songions. Le calcul ainsi conduit[1] n'exige que 4 multiplications au lieu des 10 qu'imposerait une méthode inspirée à la lettre de l'expression (2).

Ceci dit, il reste encore plusieurs manières assez différentes d'exprimer un algorithme, comme nous allons le voir sur des exemples.

Examinons tout d'abord le problème du rangement d'une suite finie S de n nombres dans l'ordre des valeurs croissantes. Si on fait abstraction des termes égaux de S , ce problème est celui de la génération du groupe symétrique d'ordre n . Le fait que tout sous-groupe d'un tel groupe qui contient un nombre suffisant de permutations de deux éléments est identique au groupe

---

(1) Ce procédé est connu sous le nom de "schéma" (ou algorithme) de HORNER.

lui-même, va nous fournir le principe d'une solution : on peut obnir le rangement désiré des termes de la suite S en permutant à bon escient deux éléments voisins de la suite initiale ou des suites obtenues par ce procédé. Vous remarquerez que les mots " à bon escient" font allusion à un processus ... non précisé. Une description détaillée de l'algorithme auquel nous songeons devra se montrer plus explicite.

Si nous désignons par a_1, a_2,...,a_n le premier, le second,..., le dernier élément de la suite, au moment où nous en parlons, nous pouvons dire par exemple :

Comparer a_1 et a_2 ; si $a_1 > a_2$, permuter a_1 et a_2 , sinon :
Comparer a_2 et a_3 ; si $a_2 > a_3$, permuter a_2 et a_3 et reprendre au début ; sinon :
Comparer a_3 et a_4 ; si $a_3 > a_4$, permuter a_3 et a_4 et reprendre au début ; sinon :
Comparer a_4 et a_5 ; etc...

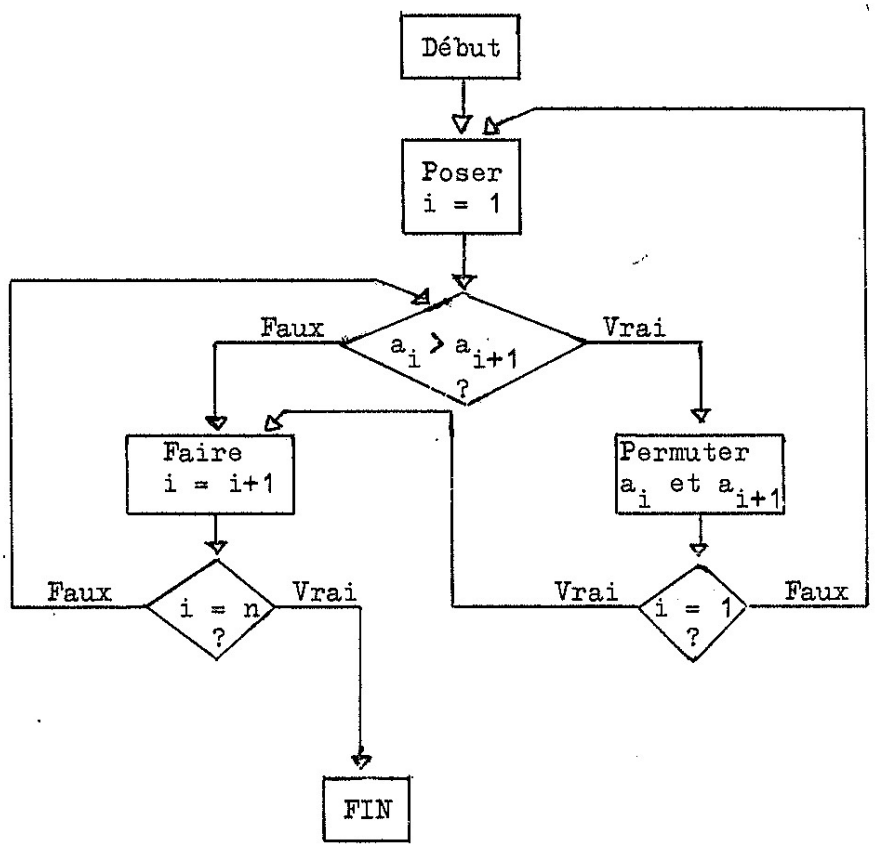
La marche à suivre est parfaitement claire pour qui sait, non seulement "lire, écrire et compter" , mais encore prendre conscience de la progression des indices et ... s'arrêter à temps. On peut être un peu plus explicite encore en disant, par exemple

Comparer a_1 et a_2 ; si $a_1 > a_2$, permuter a_1 et a_2 , sinon :
Si $2=n$[(1)], terminé ; sinon :
Comparer a_2 et a_3 ; si $a_2 > a_3$, permuter a_2 et a_3 et reprendre au début ; sinon :
Si $3=n$, terminé ; sinon :
Comparer a_3 et a_4 ; si $a_3 > a_4$, permuter a_3 et a_4 et reprendre au début ; sinon :
Si $4=n$, terminé ; sinon :
Comparer a_4 et a_5 ; etc...

Ainsi,nous ne supposons, chez l'auditeur, que la faculté d'apercevoir la progression des indices, autrement dit, de généraliser. Ce n'est pas excessif si l'auditeur est un être intelligent; c'est

(1) Ceci suppose que la suite initiale a au moins 2 éléments.

trop si l'interlocuteur est une machine. C'est pourquoi, à son intention, nous dessinerons le schéma suivant :



Le symbolisme -que nous adopterons dorénavant en le complétant au besoin - est facile à deviner :

à partir du "Début" et jusqu'à la "Fin"

nous effectuons une suite d'opérations décrites explicitement et exprimés dans un cartouche :

 , s'il s'agit d'une opération "arithmétique" (par exemple :"poser i = 1" c'est à dire attribuer à l'indice i la valeur 1) ou d'un algorithme très simple (par exemple : "permuter a_i et a_{i+1}");

 , s'il s'agit d'une comparaison (par exemple : "$a_i > a_{i+1}$?" - c'est à dire : a_i est-il supérieur à a_{i+1} ?).

L'enchaînement de ces opérations, leur succession, est indiqué par des flèches :

Vous remarquerez que, sous peine d'indétermination,

- une flèche seulement peut lier l'opération indiquée dans un  à la suivante ;

- deux flèches et deux seulement doivent lier la comparaison repérée par un  aux opérations suivantes : l'une d'elle porte la mention "Vrai" - ou encore "oui" - l'autre la mention "Faux" - ou encore "non" - : elles indiquent la marche à suivre lorsque la condition est vérifiée ou ne l'est pas.

Par contre, plusieurs flèches peuvent parfaitement renvoyer à la même opération.

Ces remarques étant faites, la signification du schéma ci-dessus devient tout à fait claire [1] : c'est une manière explicite d'exposer l'algorithme qui fournit la solution de notre problème. On donne d'ordinaire à un tel schéma le nom d'organigramme .

L'organigramme est à l'expression en langage usuel de l'algorithme ce que l'affiche est au placard publicitaire, ce que le dessin humoristique est à l'histoire drôle : il s'adresse à un spectateur plutôt qu'à un auditeur. Il peut donc, chez certains sujets, faciliter grandement la compréhension d'un algorithme.

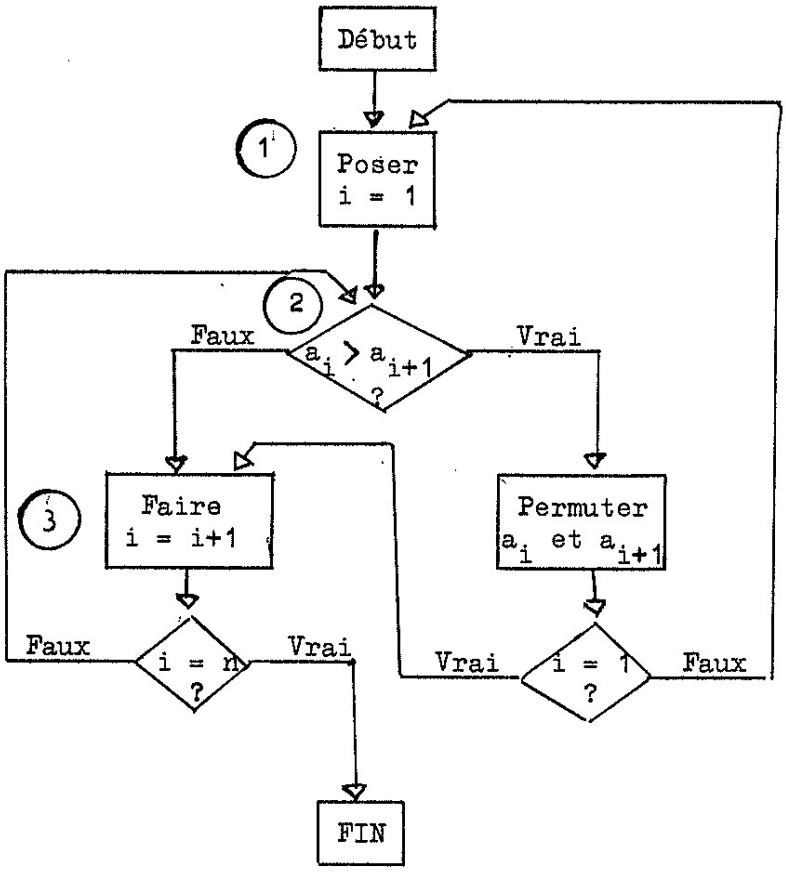
Mais il se trouve que les machines s'accomodent mieux d'un langage. Qu'à cela ne tienne, nous allons "traduire" notre organigramme, sinon en "bon" français, du moins en un langage compréhensible, et revenir, à peu de chose près, à notre première rédaction, en explicitant cette fois le contenu du mot "etc" .

Rien n'est plus simple si on prend la précaution de repérer, dans l'organigramme, au moins celles des opérations auxquelles renvoient plusieurs flèches, en utilisant par exemple des groupes de lettres

---

(1) Remarquons tout de même que l'opération "Faire i=i+1" signifie: "attribuer à l'indice i sa valeur précédente augmentée de 1", autrement dit, "augmenter de 1 la valeur de l'indice i ".

ou encore des nombres entiers cardinaux (car l'ordre dans lequel on les emploie est indifférent).



Début

1 : Poser i = 1 ;

2 : Si $a_i > a_{i+1}$ alors { permuter a_i et a_{i+1} ; si i=1 aller à 3 sinon aller à 1 }

sinon

3 : Faire i = i+1 ; si i = n terminé, sinon aller à 2 .

Que ce texte exprime exactement le contenu de l'organigramme, cela n'est pas douteux. Ce qui l'est peut-être un peu plus c'est qu'il soit une expression correcte de l'algorithme qui doit nous permettre de ranger dans l'ordre des valeurs croissantes les termes d'une suite quelconque $a_1, \ldots, a_n$ (pourvu que n soit $\geqslant 2$).

Pour en avoir le coeur net nous allons prendre un exemple, c'est-à-dire appliquer l'algorithme au rangement des termes d'une suite particulière (n=3):

a_1,	a_2,	a_3
8,	9,	3

Appliquer l'algorithme à cette suite, cela revient à effectuer les opérations suivantes

on pose i = 1 ; $a_1 \leqslant a_2$ (car $8 \leqslant 9$) donc,

on pose i = 2 ; $2 \neq n$; $a_2 > a_3$ (car $9 > 3$) donc

on permute a_2 et a_3 :

a_1	a_2	a_3
8	3	9

comme $i \neq 1$,

on pose i = 1 ; $a_1 > a_2$ (car $8 > 3$) donc

on permute a_1 et a_2 :

a_1	a_2	a_3
3	8	9

comme i = 1 ,

on pose i = 2 ; $2 \neq n$; $a_2 \leqslant a_3$ (car $8 \leqslant 9$) donc,

on pose i = 3 ; 3 = n . Fin .

L'opération est terminée.

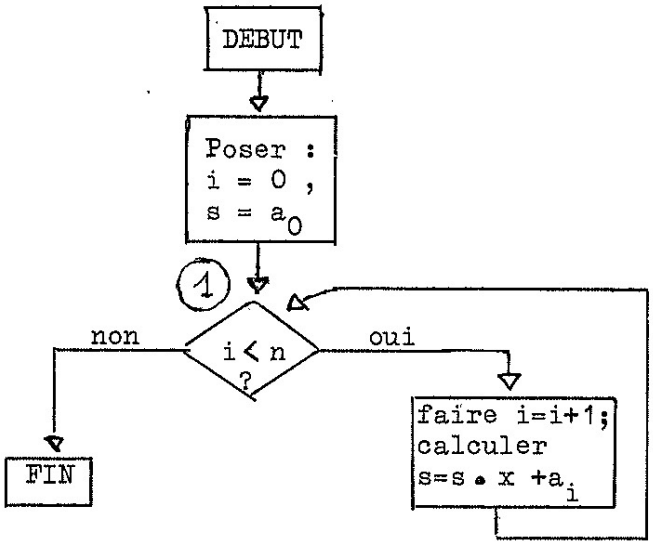
On constate que les termes de la dernière suite obtenue sont bien rangés dans l'ordre des valeurs croissantes ($a_i \leqslant a_{i+1}$ pour $i = 1,\ldots,n-1$) .

Peut-on conclure que notre algorithme fournit dans tous les cas la solution du problème ? Contentons-nous de remarquer que notre exemple était tel que nous avons parcouru, en le traitant, toutes les branches de l'organigramme.

Pour nous familiariser avec ces modes d'expression des algorithmes, nous allons examiner quelques autres problèmes, et tout d'abord celui du calcul de la valeur de l'expression

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$$

le schéma de Horner peut se traduire dans l'organigramme suivant :



En langage courant :

Début

Poser $i = 0$, $s = a_0$;

1 : si $i < n$ alors { faire $i=i+1$; calculer $s=s \cdot x+a_i$; aller à 1 }
sinon, terminé .

Ainsi, pour calculer la valeur de l'expression

$$x^2 + 2x + 3$$

pour la valeur -2 de x, on exécutera les calculs suivants

Poser $i = 0$, $s = 1$;

$0 < n$, donc faire $i = 1$ et $s = 1 \times (-2)+2 = 0$;

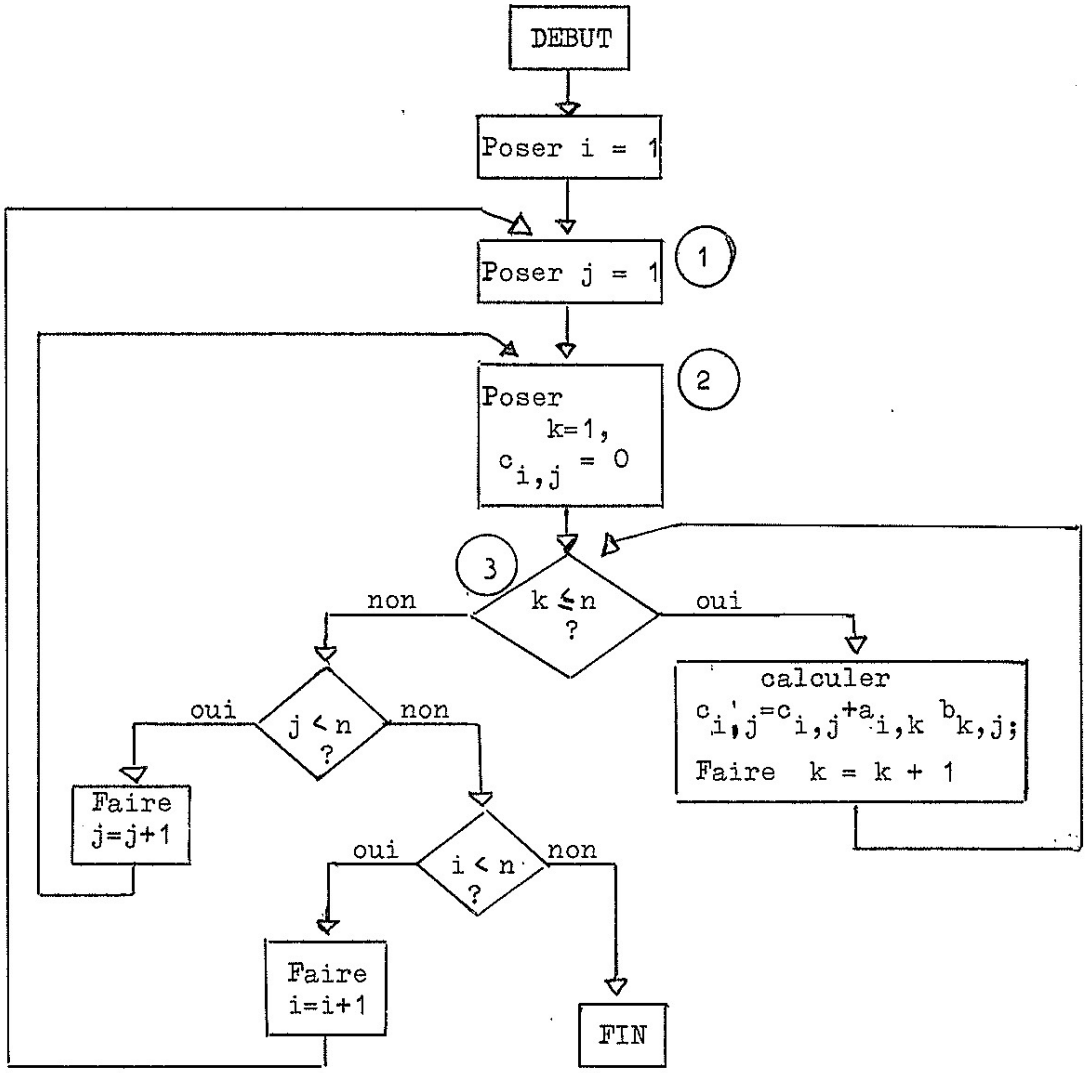
$1 < n$, donc faire $i = 2$ et $s = 0 \times (-2)+3 = 3$

$2 \geqslant n$, terminé.

La valeur finale de s est la valeur cherchée.

Reprenons maintenant le produit de deux matrices carrées A et B, d'ordre n , à éléments réels. Soit $C = A \times B$ donc

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} \times B_{k,j}$$



En langage courant :

Début

Poser i = 1 ; 1 : Poser j=1 ; 2: Poser k=1 , $c_{i,j}$ = 0 ;

3: si k ≤ n alors { Calculer $c_{i,j}=c_{i,j}+a_{i,k} \times b_{k,j}$; Faire k=k+1; aller à 3 }

sinon

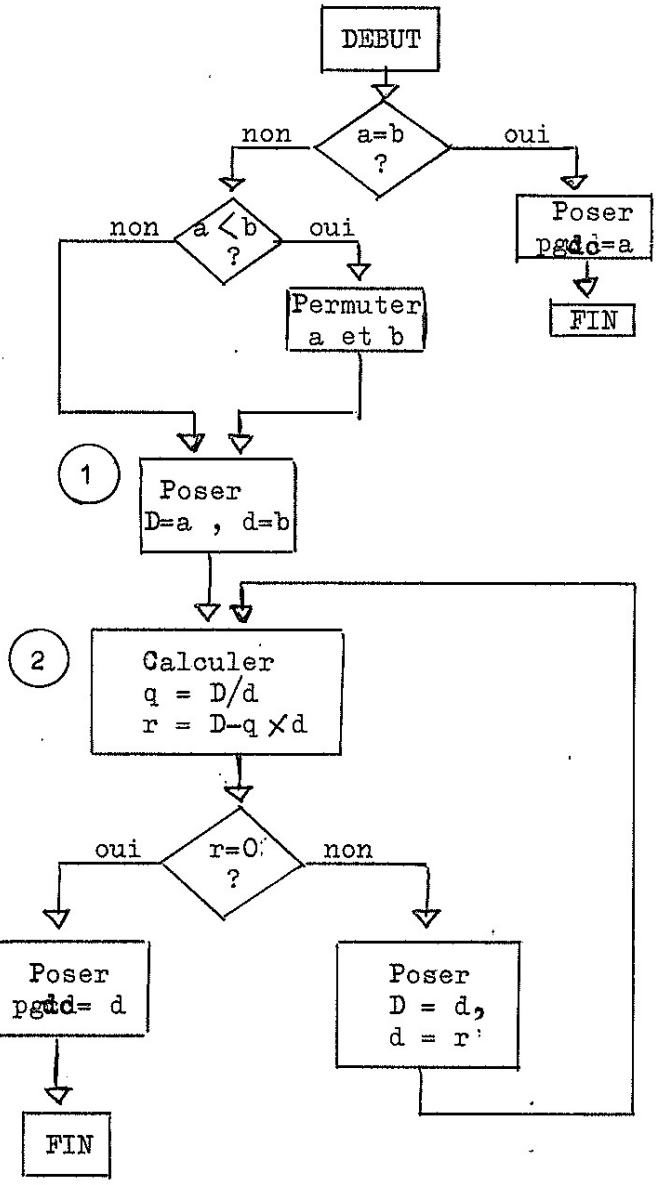
si j < n alors { Faire j = j+1 ; aller à 2 }

sinon

si i < n alors { Faire i = i+1 ; aller à 1 }

sinon, terminé.

Enfin l'algorithme d'Euclide pour le calcul du pgcd de deux nombres a et b peut s'exprimer de la façon suivante :



En langage courant :

Début

Si a = b alors {poser pgdd = a , terminé} sinon

Si a < b alors permuter a et b ;

1: Poser D = a , d = b ;

2: Calculer q = D/d , r = D - q×d ;

Si r = 0 alors {poser pgdd = d , terminé} sinon

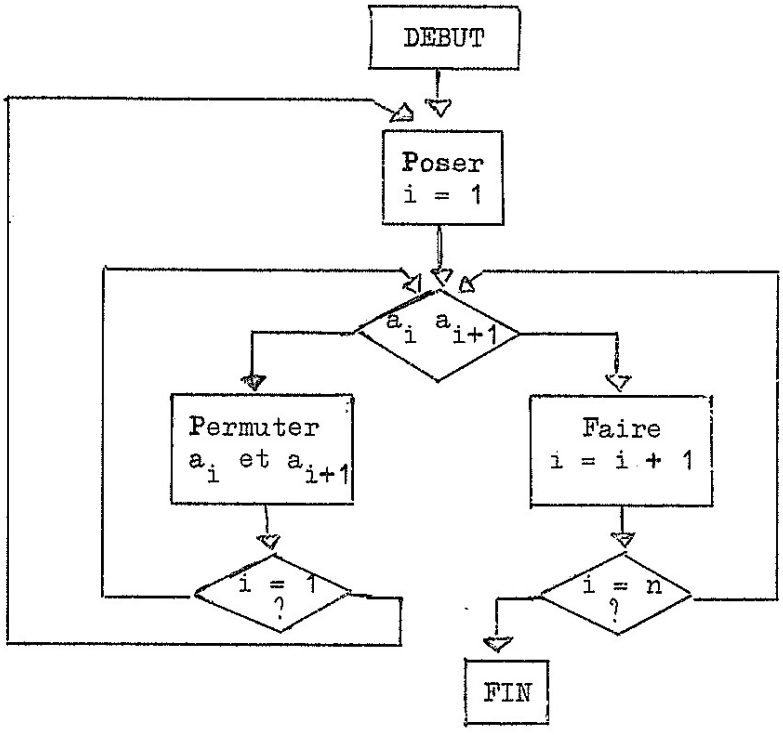
Poser D = d , d = r et aller à 2 .

EXERCICE III.1

Vérifier, sur des exemples simples, les deux algorithmes précédents.

EXERCICE III.2

Compléter l'organigramme suivant de façon à ce qu'il exprime un algorithme pour le rangement d'une suite de nombres par ordre de grandeurs croissantes



EXERCICES III.3,4,5

Exprimer sous forme d'organigramme et en langage courant des algorithmes qui permettent de résoudre les problèmes suivants :

3. Calcul de la dérivée du polynôme

$$a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$$

4. Résolution de l'équation du second degré.

5. Calcul de $\int_a^b 1/(1+x^2)\,dx$ par la méthode des trapèzes.

EXERCICES III.6

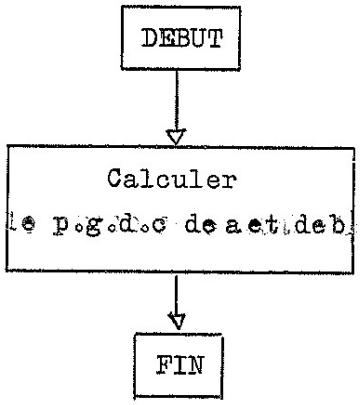
Améliorer l'algorithme de rangement des nombres par ordre de grandeur croissante en généralisant la méthode utilisée dans l'exemple ci-dessous

termes considérés	suite	a_1	a_2	a_3	a_4	a_5	a_6
		2	1	5	4	3	6
a_1 et a_2		1	2	5	4	3	6
a_2 et a_3		-	-	-	-	-	-
a_3 et a_4		1	2	4	5	3	6
a_2 et a_3		-	-	-	-	-	-
a_4 et a_5		1	2	4	3	5	6
a_3 et a_4		1	2	3	4	5	6
a_2 et a_3		-	-	-	-	-	-
a_5 et a_6		-	-	-	-	-	-

terminé

IV - LA NOTION D'"OPERATION ELEMENTAIRE" ET LA REPRESENTATION DES NOMBRES

Nous avons dit plus haut (1) que "résoudre automatiquement un problème" c'était, finalement, le soumettre à un calculateur capable d'effectuer un petit nombre d'opérations "simples". C'est cette idée d'opération "simple" ou encore "élémentaire" que nous allons préciser maintenant. Car enfin, suivant notre penchant naturel à la paresse - ou sacrifiant à certaines conventions - nous pourrions être tentés de résumer l'algorithme d'Euclide dans l'organigramme suivant :



Ce qui est admirable c'est que cela réussisse parfois: certaines machines savent en effet calculer le p.g.d.c de deux nombres (1). Mais la plupart du temps il n'en est rien: une machine déterminée ne sait effectuer qu'un nombre fini, très petit (3), d'opérations, celles qui sont définies dans son"mode d'emploi".

Nous allons le montrer sur un exemple, celui des machines de Turing (4).

---

(1) Page 9 .
(2) Ceci, grâce à un "sous-programme".
(3) 5, 50, 100, 200; au delà, la méfiance s'impose.
(4) L'exposé qui suit n'est pas une version orthodoxe de la théorie des machines de Turing. L'essentiel, pourtant, a été conservé; on le verra aisément en se reportant, par exemple, à l'ouvrage de Martin DAVIS: Computability and unsolvability , Mc GRAW-HILL 1958.

Une telle machine se compose essentiellement d'un mécanisme capable d'effectuer les opérations qu'on va énumérer dans un instant, quand on lui fournit une certaine "information".

Cette information, c'est le contenu d'une bande de papier indéfiniment prolongeable dans les deux sens et divisée en cases juxtaposées :

le contenu de chacune de ces cases peut être

soit blanc (symbolisé par B)
soit noir (symbolisé par N)

Une "tête de lecture et d'écriture" se trouve, à tout moment, en face d'une case et d'une seule; on la représente schématiquement par le symbole △ placé au dessous de cette case :

Le contenu de la bande et la position de △ par rapport à elle, à un moment donné, constituent une "configuration".

Dans tous les cas intéressants, les configurations qui ne diffèrent que par le contenu de cases suffisamment éloignées de △ sont équivalentes.

Les nombres entiers $\geqslant 0$ sont représentés de la façon suivante :

0 : 

1 : 

2 : 

etc.

A partir d'une configuration initiale la machine peut effectuer

un certain nombre d'opérations élémentaires; chacune d'elles fait l'objet d'un ordre ou instruction , donné en temps voulu à la machine. Une suite finie d'instructions constitue un programme ; un programme exprime - en général - un algorithme bien défini.

Les seules instructions dont on dispose pour écrire les programmes sont les suivantes:

1. Examiner le contenu de la case explorée; s'il est noir aller à l'instruction α , sinon aller à l'instruction β du programme.

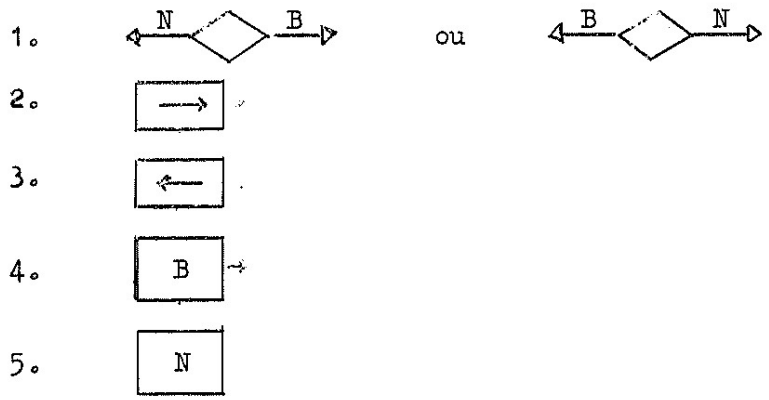
2. Déplacer la tête de lecture pour explorer la case immédiatement à droite; puis exécuter l'instruction suivante.

3. Déplacer la tête de lecture pour explorer la case immédiatement à gauche; puis exécuter l'instruction suivante.

4. Inscrire B dans la case explorée (après avoir effacé le contenu précédent); puis exécuter l'instruction suivante.

5. Inscrire N dans la case explorée (après avoir efface le contenu précédent); puis exécuter l'instruction suivante.

Pour pouvoir traduire facilement un algorithme par un organigramme on adoptera le symbolisme suivant :



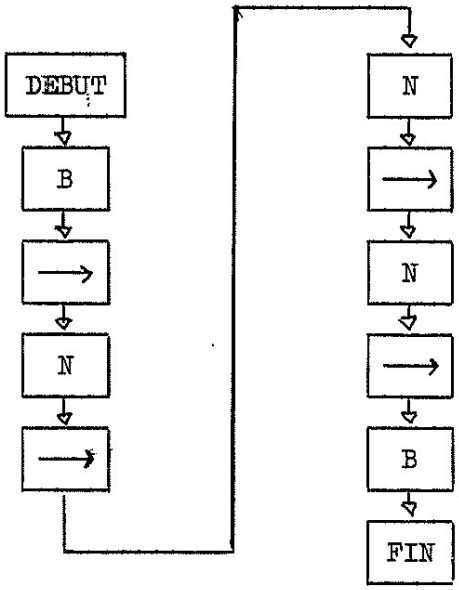
Aussi surprenant qu'il puisse paraître, une telle machine est capable de calculer la valeur de n'importe quelle fonction arithmétique lorsqu'on lui fournit la valeur de ses arguments.(1)

Bien entendu il faut, pour cela, lui fournit un algorithme approprié. Nous allons en voir quelques exemples.

---

(1) Voir Martin DAVIS op. cit.

Pb1. Ecrire le nombre 2 :(1)



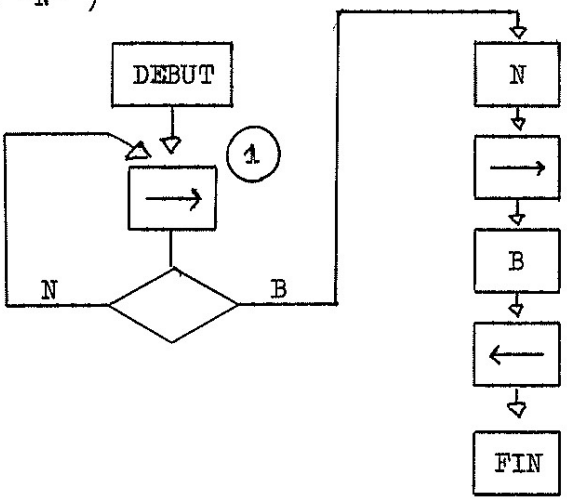
ou encore :

début ; B ; ⟶ ; N ; ⟶ ; N ⟶ ; N ; ⟶ ; B ; fin

Quelle que soit la configuration initiale on obtiendra, à la fin du calcul, une configuration qui sera caractérisée, au voisinage de Δ par

... B N N N B ...
Δ

Pb2. Ajouter 1 à un nombre (on suppose qu'au début Δ est au dessous du premier "N" et qu'à la fin Δ est au dessous du dernier "N")



(1) DEBUT et FIN ne sont pas, à proprement parler, des "opérations" mais des indications ... utiles.

ou encore :

Debut ; 1 : ⟶ ; si N aller à 1 sinon N ; ⟶ ; B ; ⟵ ; fin.

ou encore :

Début ; tant que le contenu de la case explorée est N regarder à droite; (lorsque le contenu de la case explorée est B) N ; ⟶ ; B ; ⟵ ; fin .

Exemple :

Pour former 3 à partir de 2 , c'est à dire pour passer de la configuration

B N N N B
  △

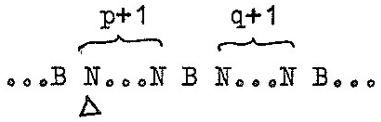
à la configuration

B N N N N B
        △

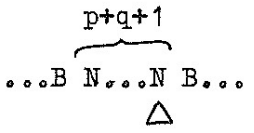
la machine effectuera les opérations suivantes :

⟶ ; ⟶ (car la case explorée contient N) ;
⟶ (car N) ; N (car B) ; ⟶ ; B ; ⟵

Pb3. Calculer la somme de deux nombres. Plus précisément : passer la configuration

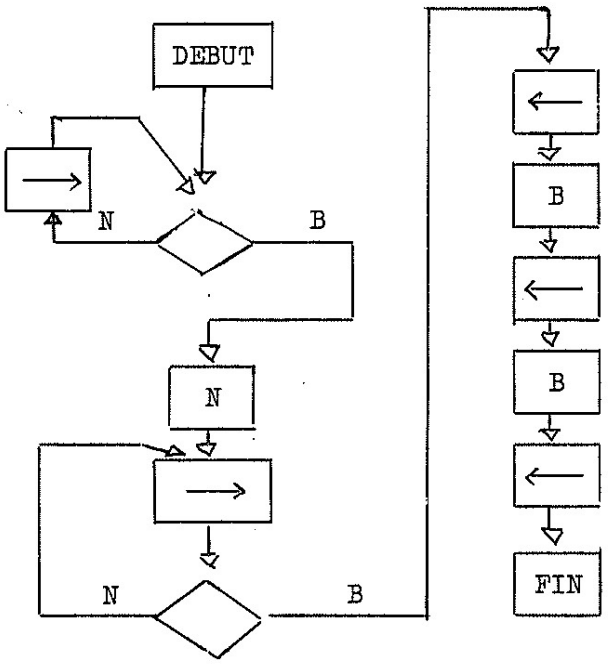


à la configuration

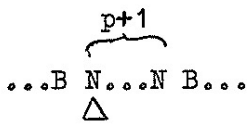


Tant que le contenu de la case explorée est N regarder à droite; (dès qu'il est B) écrire N puis
regarder à droite tant que le contenu de la case explorée est N; (dès qu'il est B) regarder à gauche; écrire B ; regarder à gauche ; écrire B ; regarder à gauche.

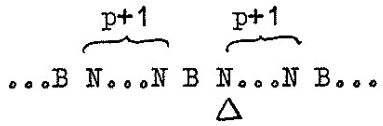
ou encore :



Pb4. Ecrire une seconde fois un nombre donné . Plus précisément, passer de la configuration



à la configuration



Désignons par e le contenu de la case explorée à un moment-donné. L'algorithme utilisé peut s'exprimer ainsi :

début

tant que e = N , regarder à droite ;
(lorsque e = B) regarder à droite ;
1: N ; → ; B ; ← ;
tant que e = N , regarder à gauche ;

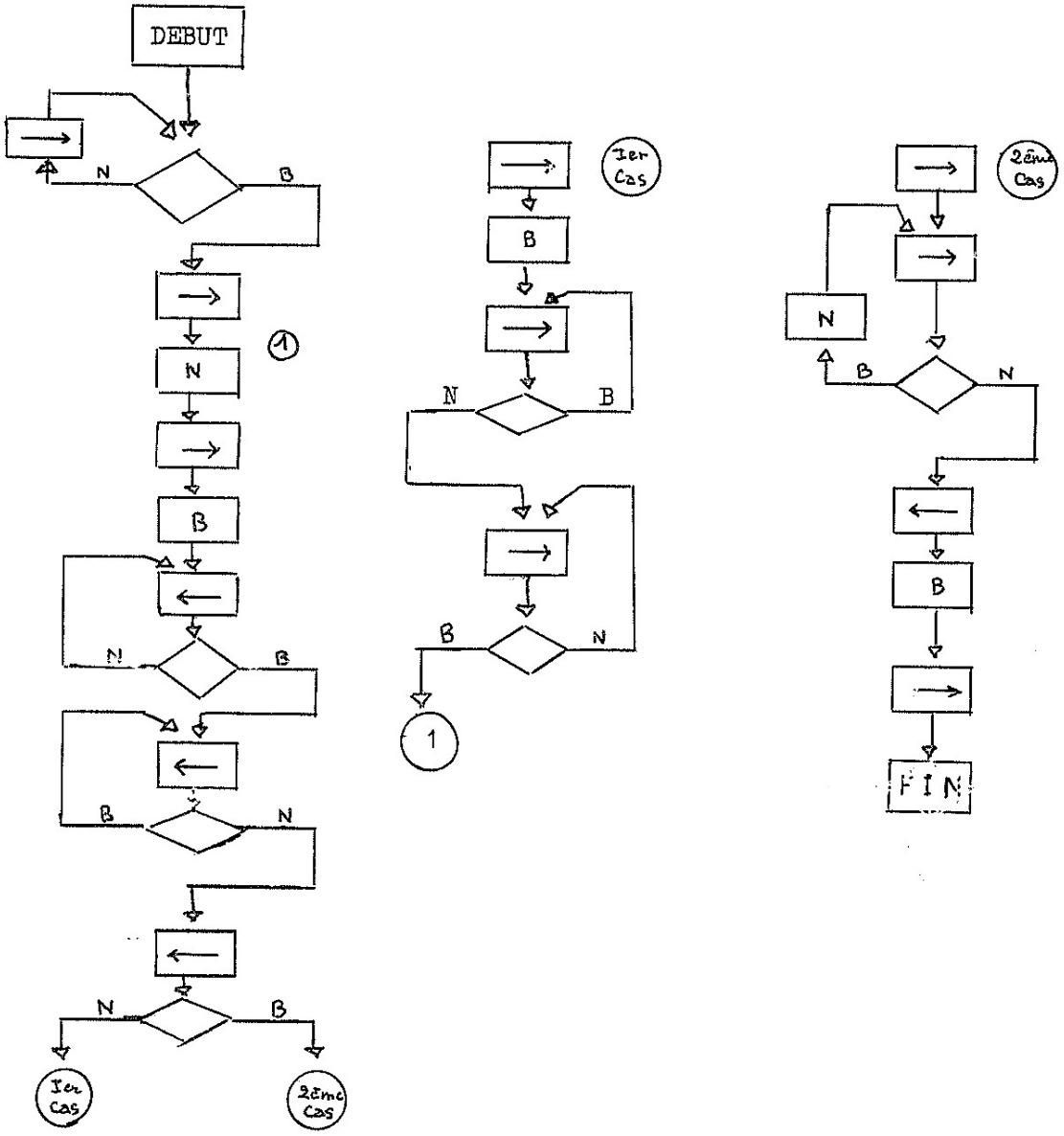
(lorsque e = B) regarder à gauche ;
tant que e = B regarder à gauche ;
(lorsque e = N) regarder à gauche ;
si e = N aller à "Ier cas" , sinon à "2ème cas" ;

Ier Cas: →; B ; tant que e = B , regarder à droite puis tant que e = N regarder à droite ;
(lorsque e = B) aller à 1 .

2ème Cas: →; →; tant que e = B , écrire N et regarder à droite
(lorsque e = N) ← ; B ; →;

fin.

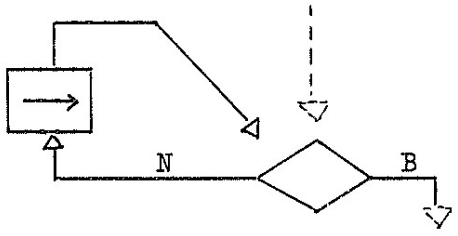
ou encore



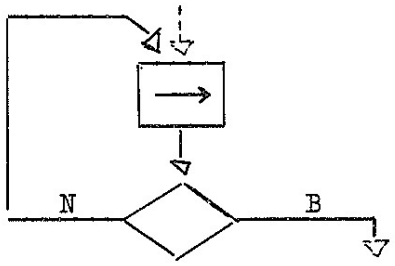
Ces quelques exemples suffisent pour faire comprendre ce qu'est une opération élémentaire(1) exécutable par une machine. Ils permettent déjà d'entrevoir ce que peut être, pour une machine, une opération "complexe" ou "synthétique"(2). Ainsi l'opération

"tant que le contenu de la case explorée est N regarder à droite"

n'est rien d'autre que l'enchaînement d'opérations élémentaires



enchaînement qui est équivalent au suivant



lorsque la configuration de départ est de la forme

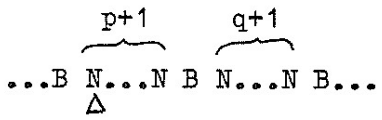
... N ...

(1) Exprimée par une "micro-instruction".

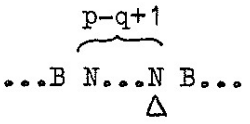
(2) Exprimée par une "macro-instruction".

EXERCICE IV.1

Décrire un algorithme permettant à une machine de Turing de passer de la configuration



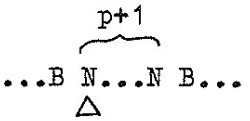
(où $p \geqslant q$), à la configuration



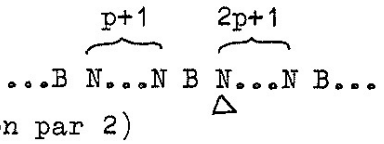
(soustraction)

EXERCICE IV.2

Décrire un algorithme permettant à une machine de Turing de passer de la configuration



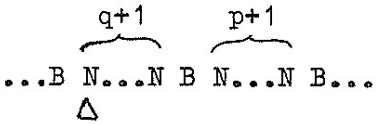
à la configuration



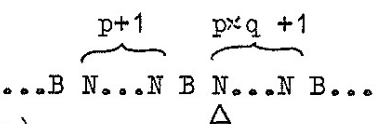
(multiplication par 2)

EXERCICE IV.3

Décrire un algorithme permettant à une machine de Turing de passer de la configuration



à la configuration



(multiplication)

T A B L E D E S M A T I E R E S