INFORMATIQUE THÉORIQUE ET APPLICATIONS

J.-C. FOURNIER

Pour en finir avec la dérécursivation du problème des tours de Hanoï

Informatique théorique et applications, tome 24, n° 1 (1990), p. 17-35

<http://www.numdam.org/item?id=ITA_1990__24_1_17_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 24, n° 1, 1990, p. 17 à 35)

POUR EN FINIR AVEC LA DÉRÉCURSIVATION DU PROBLÈME DES TOURS DE HANOÏ (*)

par J.-C. FOURNIER (1)

Communiqué par J.-E. PIN

Résumé. – *Nous discutons de façon approfondie la dérécursivation du classique problème des Tours de Hanoï et nous donnons une formule de récurrence nouvelle et fondamentale qui permet de retrouver les propriétés itératives connues et de proposer un principe très simple de solution itérative. Finalement, on observe que, pour ce problème des Tours de Hanoï, l'itération conserve quelque chose de la récursivité et qu'ainsi ce problème apparaît bien comme de nature foncièrement récursive.*

Abstract. – *We discuss in a detailed way how to develop the classical Towers of Hanoï problem in a non recursive form, and we give a new and fundamental formula of recurrence which allows us to rediscover the known iterative properties and to propose a very simple principle of iterative solution. Finally, we observe that, for this Towers of Hanoï problem, the iteration preserves something of the recursion and thus this problem appears indeed of fundamentally recursive nature.*

1. INTRODUCTION

Tout le monde connaît le problème des Tours de Hanoï : On a 3 piquets et un certain nombre de disques de diamètres différents enfilés par ordre de diamètres décroissants sur l'un des piquets. Il s'agit de transporter ces disques sur un autre piquet en ne déplaçant qu'un seul disque à la fois d'un piquet à un autre et en respectant la règle qu'un disque ne doit jamais être placé sur un disque de diamètre inférieur. En outre on cherche à minimiser le nombre de déplacements de disques.

La solution générale, en $2^n - 1$ déplacements de disques pour n disques, est bien connue (cf. *fig.* 1 pour le cas $n = 3$). Elle est souvent présentée dans les ouvrages d'Algorithmique et de Programmation comme l'archétype de la

(*) Reçu janvier 1988, version finale juin 1989.
(1) Université Paris - Val de Marne, Faculté de Sciences Économiques et de Gestion, 58, avenue Didier, 94210 La-Varenne-Saint-Hilaire, France.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754 90/01 17 19/$3.90/

récursivité. Mais de nombreux auteurs ont proposé des stratégies itératives de solutions qui les ont amenés à contester la nature véritablement récursive du problème. Par exemple Arsac [1, 2] qui, avec d'autres, observe que le plus petit disque doit être déplacé un coup sur deux et toujours dans le même sens cyclique des piquets (numérotés 1, 2, 3), et que les autres coups sont déterminés par la règle du jeu. Ou encore Lavallée [7] : la suite des sommes des numéros des piquets concernés par chaque déplacement est périodique de période 3 (par exemple pour $n=3$ on a la suite : 3, 4, 5, 3, 4, 5, 3), ce qui permet de prévoir chaque coup (on connaît les piquets concernés, le sens de déplacement est forcé par la règle du jeu).

Ces stratégies, trouvées en observant la solution, suppose connu l'état du jeu. On en a proposé d'autres pour lesquelles ce n'est pas nécessaire mais qui font alors souvent intervenir le nombre de zéros à droite de la représentation binaire du coup. Meyer-Baudoin [8] qualifie ceci de curiosité mathématique et Arsac [2] se demande comment amener de façon naturelle ce nombre. Question d'autant plus cruciale que dans cette dernière référence le problème des Tours de Hanoï est une illustration significative des techniques de dérécursivation et de transformations de programmes. Lavallée [7] donne une stratégie différente qui ne demande pas non plus de connaître l'état du jeu, mais avec des formules compliquées (théorème 3 dans [7]). Hardouin Duparc [5] propose également une telle solution, en utilisant une pile d'automates.

Dans l'abondante littérature sur le problème des Tours de Hanoï, les stratégies données ne sont pas toujours très clairement ou même très correctement justifiées. Ce qui amène Er [4], s'inspirant de ce qu'on fait en Intelligence Artificielle, à proposer une représentation selon lui mieux adaptée et qui simplifie la justification de propriétés habituellement considérées pour résoudre le problème, comme par exemple celle déjà citée : le plus petit disque est déplacé toujours dans le même sens cyclique, propriété nullement évidente.

Dans ce qui suit nous nous proposons de montrer :

1° que les solutions itératives que nous venons d'évoquer se déduisent en fait très simplement et très naturellement de la solution récursive classique. Ceci par une dérécursivation *optimisée* et sans qu'il soit besoin de considérer d'autre représentation que celle, classique aussi, de l'arborescence des appels récursifs. Il y a quand même à mettre en œuvre une technique d'inversion des fonctions donnant les arguments lors des appels récursifs, ce qui est ici fondamental et parfois méconnu. Ce premier point relève d'une mise au point du problème des Tours de Hanoï ayant un caractère surtout didactique.

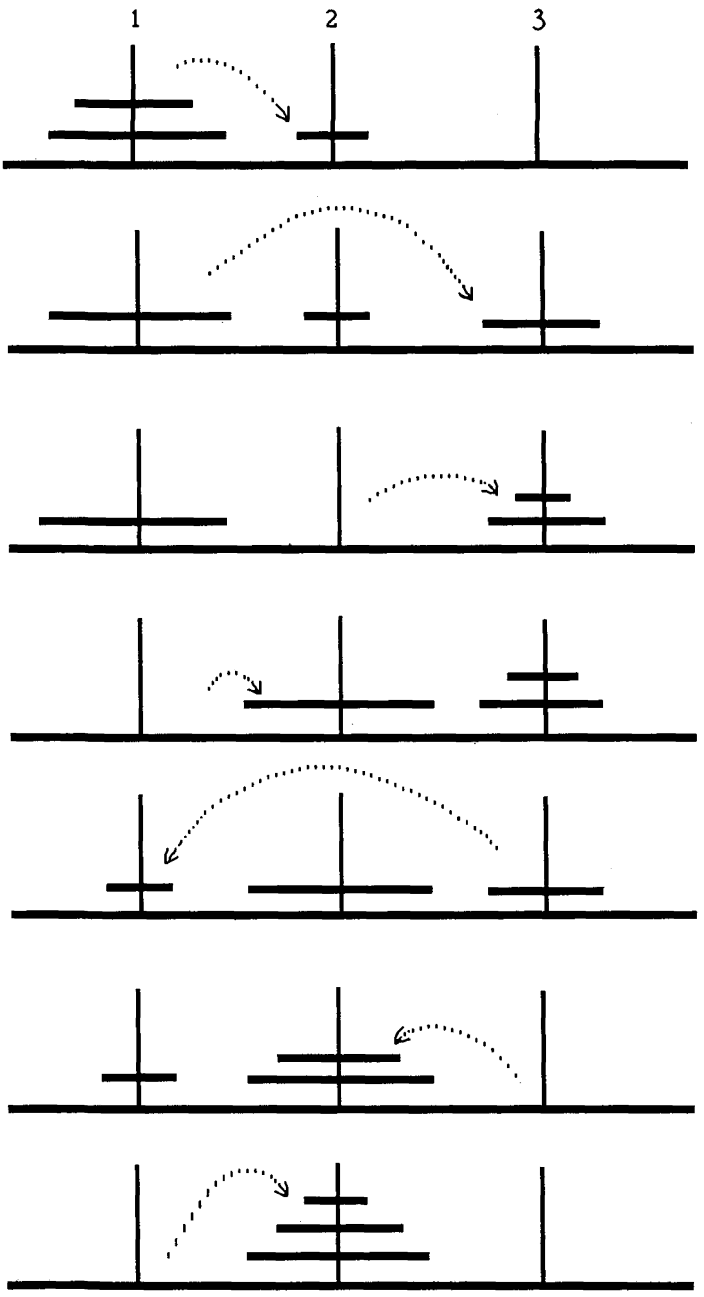


Figure 1

2° qu'il existe une formule de récurrence remarquablement simple, et apparemment encore inédite, qui permet de retrouver toutes les solutions itératives simples évoquées. Cette formule montre qu'en fait la solution du problème n'est que la répétition des 3 premiers déplacements à une éventuelle inversion près des piquets concernés, inversion complètement définie.

3° que dans les solutions itératives les plus simples il reste quelque chose de la récursivité, sous la forme ici d'un compteur binaire qui en fait simule une pile de dérécursivation. Ce qui montre l'ambiguité de parler d'élimination de la récursivité dès lors qu'on se passe, formellement, d'une pile de dérécursivation. En fait, c'est plutôt sur l'existence d'une solution purement itérative, ou même simplement sur la signification d'une telle dénomination, qu'il faudrait s'interroger pour le problème des Tours de Hanoï, comme sans doute d'ailleurs aussi pour bien d'autres problèmes.

2. SOLUTION RÉCURSIVE ET PREMIÈRE DÉRÉCURSIVATION CLASSIQUE

Dans tout ce qui suit les disques sont supposés numérotés de 1 à n, du plus petit au plus grand. Les piquets sont numérotés de 1 à 3. Étant donnés p et q deux numéros de piquets distincts, on pose $r=6-p-q$ qui représente le numéro du troisième piquet, autre que les piquets p et q.

Le programme récursif suivant est la solution classique du problème des Tours de Hanoï. La procédure déplacer (i, p, q) donne les déplacements à effectuer pour le transport des disques numérotés de 1 à i du piquet p au piquet q en se servant du piquet intermédiaire r.

```
programme Hanoï;
procédure déplacer (i, p, q);
début
  si i=1 alors
    afficher (i ' de ' p ' à ' q);
  sinon
    déplacer (i-1, p, r);
    afficher (i ' de ' p ' à ' q);
    déplacer (i-1, r, q)
  fin si
fin déplacer;
{*programme principal*}
début
  lire (n);
  déplacer (n, 1, 2)
fin.
```

Une première dérécursivation « brute », classiquement faite à l'aide d'une pile, conduit au programme suivant. On observe que le 2e appel récursif étant

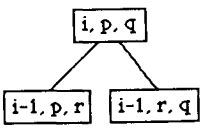
terminal il est inutile d'empiler les données au moment de cet appel, puisque celles-ci ne seront plus utilisées.

```
debut
  lire (n);
  i:=n; p:=1; q:=2;
  pile:=∅;
1 : si i=1 alors
      afficher (i ' de ' p ' à ' q);
  sinon
      empiler (i, p, q); i:=i-1; q:=r; aller en 1;      (*1er appel récursif*)
2 : afficher (i ' de ' p ' à ' q);
      i:=i-1; p:=r; aller en 1;      (*2e appel récursif*)
  fin si;
  si pile ≠ ∅ alors
      dépiler (i, p, q); aller en 2      (*retour récursif*)
  fin si
fin.
```

Malgré une apparente simplicité, le fonctionnement de ce programme est relativement subtil : tant qu'on sort du 2e appel récursif il n'y a rien à faire d'autre que de remonter à la procédure appelante. C'est seulement lorsqu'on finit par sortir du 1er appel (qui n'est pas terminal) qu'on a à dépiler puis à se brancher en 2 (suite du 1er appel dans le programme). C'est pourquoi le retour récursif se fait directement en dépilant et en allant en 2 (inutile de s'occuper des retours du 2e appel).

3. PARCOURS DE L'ARBORESCENCE DES APPELS RÉCURSIFS

On peut visualiser très bien le fonctionnement du programme itératif précédent en considérant ce qu'on appelle l'arborescence des appels récursifs. Par exemple ici la procédure déplacer (i, p, q) appelle la procédure déplacer $(i-1, p, r)$ puis la procédure déplacer $(i-1, r, q)$, ce qu'on représente par le diagramme ci-contre, où chaque case contient les données de l'appel correspondant.



L'ensemble des appels peut ainsi être représenté par une arborescence binaire (figure 2 pour le cas $n=4$).

Il est facile de voir que le fonctionnement du programme précédent correspond à un *parcours symétrique* de cette arborescence. On trouve ainsi, dans l'ordre voulu, les actions d'affichage qui expriment la solution : il y a un affichage entre 2 appels récursifs, ce qu'on représente sur la figure par un petit trait en chaque sommet de l'arborescence entre 2 arcs (cf. *fig.* 2).

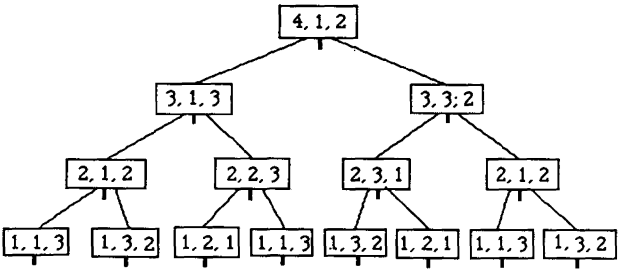
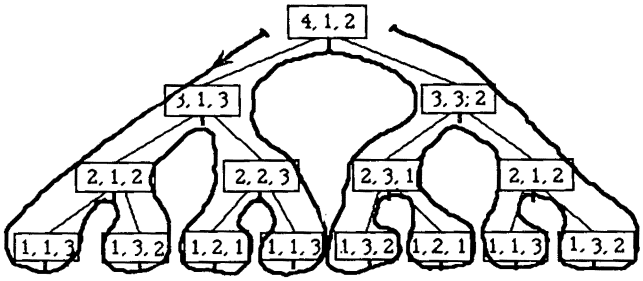


Figure 2

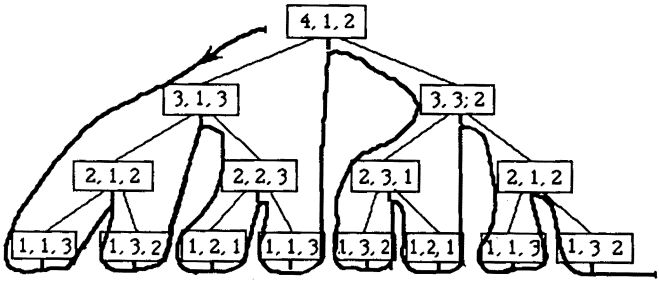
Avec une telle représentation, il devient clair qu'une dérécursivation la meilleure possible, optimisée, est celle qui consiste à aller directement d'une action d'affichage à la suivante dans l'ordre précédent, en court-circuitant les sommets de l'arborescence pour lesquels il n'y a pas d'action à accomplir.

Le programme dérécursivé précédent réalise au moins en partie ceci, en suivant la règle suivante : chaque fois qu'on prend un fils gauche (correspondant au 1er appel récursif) empiler le sommet qu'on quitte, mais pas pour un fils droit; pour remonter, dépiler. Ainsi, par exemple, on remonte directement de (1, 3, 2) à (3, 1, 3) sans passer par (2, 1, 2). On a un parcours plus court que le strict parcours symétrique, lequel correspond à un empilement des données des appels récursifs terminaux dont on sait qu'il est inutile (comparer les *fig.* 3 *a* et 3 *b*).

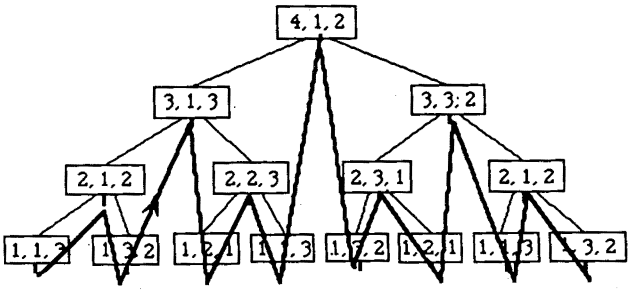
La solution serait encore meilleure, et même la meilleure possible, si on pouvait éviter par exemple de passer par (2, 2, 3) pour aller de (3, 1, 3) à (1, 2, 1), c'est-à-dire réaliser le parcours direct de la figure 3 *c*. Il faudrait pour cela, et c'est une remarque fondamentale, ne pas avoir à empiler les données lors des appels récursifs initiaux, c'est-à-dire au fond se passer complètement de la pile. Or précisément cela est possible ici du fait que *les fonctions donnant les arguments des appels récursifs sont inversibles*, ce qui permet au retour de l'appel récursif pour restaurer l'état des variables de la



(*a*)



(*b*)



(*c*)

Figure 3

procédure appelante de recalculer celles-ci plutôt que de les reprendre dans une pile ([2]).

([2]) En fait d'un point de vue purement formel la différence n'est pas si grande, la pile n'étant jamais qu'une façon de définir l'inverse de la fonction en stockant de façon adéquate ses valeurs.

4. ALGORITHME GÉNÉRAL DE PARCOURS SYMÉTRIQUE

Pour mettre en application la remarque précédente, on va déterminer par un algorithme la suite des opérateurs qui s'appliquent aux données lors d'un parcours symétrique direct comme nous venons de le définir.

On va en fait se placer dans le contexte général du parcours symétrique d'une arborescence binaire complète, définie par sa racine r, sa hauteur $n-1$, et les primitives fils gauche g et fils droit d. Les fonctions g et d sont supposées inversibles, les fonctions inverses g^{-1} et d^{-1} étant définies sur l'ensemble des sommets de l'arborescence privé de la racine.

Considérons l'algorithme suivant, dans lequel s est une variable de type sommet de l'arborescence, j et k sont des variables entières et la fonction v_2 retourne l'exposant du facteur 2 dans la décomposition en facteurs premiers de l'entier donné (ici j).

$s := g^{n-1}(r)$;
pour j **de** 2 **à** $2^n - 2$ **pas** 2 **faire**
  $k := v_2(j)$;
  $s := g^{-1} \circ d^{-k+1}(s)$;
  $s := g^{k-1} \circ d(s)$
fin pour

Nous allons montrer qu'avec cet algorithme la variable s prend pour valeur successivement tous les sommets de l'arborescence *dans l'ordre infixe*. Pour montrer cela nous allons considérer la fonction ν définie sur l'ensemble des sommets de l'arborescence et associant à chaque sommet son numéro dans l'ordre infixe, et montrer qu'au cours de l'exécution de l'algorithme $\nu(s)$ prend successivement les valeurs : 1, 2, ..., $2^n - 1$. (L'arborescence étant complète et de hauteur $n-1$ a $2^n - 1$ sommets.)

Notons $h(x)$ la hauteur d'un sommet quelconque x de l'arborescence. Étant donné un sommet x non feuille de l'arborescence (c'est-à-dire ayant deux fils), les sommets qui sont pour l'ordre infixe entre x et son fils gauche $g(x)$ sont ceux de la sous-arborescence droite de $g(x)$. Celle-ci étant de hauteur $h(g(x)) - 1$, ses sommets sont au nombre de $2^{h(g(x))} - 1$. D'où on déduit :

$$\nu(x) = \nu(g(x)) + (2^{h(g(x))} - 1) + 1.$$

Soit encore :

$$\nu(g(x)) = \nu(x) - 2^{h(g(x))}. \tag{4.1}$$

Par un calcul analogue avec le fils droit $d(x)$, on trouve :

$$\nu(d(x)) = \nu(x) + 2^{h(d(x))}. \tag{4.2}$$

Nous allons exprimer la hauteur d'un sommet en fonction de son numéro dans l'ordre infixe, par la formule :

$$h(x) = v_2(\nu(x)) \quad \text{pour tout sommet } x. \tag{4.3}$$

Il est facile de vérifier (4.3) pour la racine de l'arborescence : en effet $\nu(r) = 2^{h(r)}$ et donc $v_2(\nu(r)) = h(r)$. Par ailleurs supposant (4.3) vrai pour un sommet x non feuille on a

$$\nu(x) = 2^{h(x)}(2k+1), \qquad \text{où} \quad k \in \mathbb{N},$$

par définition de $v_2(\nu(x))$, et avec (4.1) on a

$$\nu(g(x)) = 2^{h(x)}(2k+1) - 2^{h(g(x))} = 2^{h(g(x))}(4k+1)$$

[compte tenu de $h(x) = h(g(x)) + 1$]. Il résulte de cette dernière égalité que $v_2(\nu(g(x))) = h(g(x))$. On démontrerait de même que $v_2(\nu(d(x))) = h(d(x))$. Ainsi de fils en fils à partir de la racine (ou si on veut par récurrence descendante sur la hauteur), on a bien la formule (4.3) pour tout sommet.

Pour interpréter la suite de transformations subies par $\nu(s)$ dans l'algorithme considéré, on introduit les fonctions G et D qui donnent le numéro dans l'ordre infixe respectivement du fils gauche et du fils droit en fonction du numéro dans l'ordre infixe du sommet considéré. D'après (4.3) les sommets non feuille de l'arborescence sont ceux dont le numéro est pair [en effet $h(x) > 0$ équivaut à $v_2(\nu(x)) > 0$, c'est-à-dire 2 divise $\nu(x)$]. Les fonctions G et D sont donc définies sur les entiers pairs de l'ensemble $\{1, 2, \ldots, 2^n - 1\}$ par les relations suivantes :

$$G(\nu(x)) = \nu(g(x)) \qquad \text{et} \qquad D(\nu(x)) = \nu(d(x)). \tag{4.4}$$

De (4.4), (4.1) et (4.3) appliqué à $h(g(x)) = h(x) - 1$, on tire $G(\nu(x)) = \nu(x) - 2^{v_2(\nu(x))-1}$.

Cette égalité étant vraie quel que soit le sommet x non feuille, on en déduit l'expression suivant de la fonction G :

$$G(l) = l - 2^{v_2(l)-1} \quad \text{pour tout } l \in 2\mathbb{N} \cap \{1, 2, \ldots, 2^n - 1\}. \tag{4.5}$$

On obtient de la même façon l'expression suivante de la fonction D :

$$D(l) = l + 2^{v_2(l)-1} \quad \text{pour tout } l \in 2\mathbb{N} \cap \{1, 2, \ldots, 2^n - 1\}. \tag{4.6}$$

En raisonnant par récurrence sur m et en observant que $v_2(l+a2^b)=b$ lorsque l'entier a est impair et $b < v_2(l)$, on montre ensuite les formules suivantes valables pour tout entier m positif tel que G_m et D_m sont définis :

$$G^m(l)=l-(2^m-1)2^{v_2(l)-m} \quad \text{pour tout } l\in 2\mathbb{N}\cap\{1, 2, \ldots, 2^n-1\}. \quad (4.7)$$

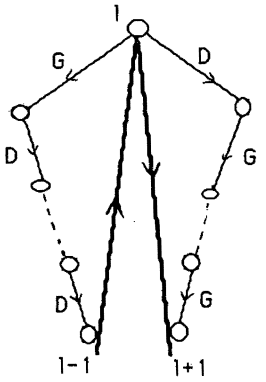
$$D^m(l)=l+(2^m-1)2^{v_2(l)-m} \quad \text{pour tout } l\in 2\mathbb{N}\cap\{1, 2, \ldots, 2^n-1\}. \quad (4.8)$$

On en déduit enfin les formules suivantes (en vérifiant que $D^{v_2(l)-1}$ et $G^{v_2(l)-1}$ sont bien définis) :

$$D^{v_2(l)-1}\circ G(l)=l-1 \quad \text{pour tout } l\in 2\mathbb{N}\cap\{1, 2, \ldots, 2^n-1\}. \quad (4.9)$$

$$G^{v_2(l)-1}\circ D(l)=l+1 \quad \text{pour tout } l\in 2\mathbb{N}\cap\{1, 2, \ldots, 2^n-1\}. \quad (4.10)$$

Remarque : Ces dernières formules, essentielles pour la suite, peuvent être retrouvées directement en considérant 3 sommets de l'arborescence consécutifs dans l'ordre infixe, de numéros l-1, l et l+1, et en observant le fait [*cf.* (4.3)] que la hauteur du sommet de numéro l est $v_2(l)$.



Revenons à l'algorithme considéré. $v(s)$ s'exprime algorithmiquement de la manière suivante :

$v(s) := v(g^{n-1}(r))$;
pour j **de** 2 **à** 2^n-2 **pas** 2 **faire**
  $k := v_2(j)$;
  $v(s) := v(g^{-1}\circ d^{-k+1}(s))$;
  $v(s) := v(g^{k-1}\circ d(s))$
fin pour

Il résulte de (4.4) (étendu aux fonctions inverses de g, d, G, D) :

$$\nu(g^{n-1}(r)) = G^{n-1}(\nu(r)),$$
$$\nu(g^{-1} \circ d^{-k+1}(s)) = G^{-1} \circ D^{-k+1}(\nu(s)),$$
$$\nu(g^{k-1} \circ d(s)) = G^{k-1} \circ D(\nu(s)).$$

Ce qui permet d'écrire l'algorithme précédent sous la forme suivante plus adaptée à l'étude de $\nu(s)$:

$\nu(s) := G^{n-1}(\nu(r))$;
pour j **de** 2 **à** $2^n - 2$ **pas** 2 **faire**
  $k := v_2(j)$;
  $\nu(s) := G^{-1} \circ D^{-k+1}(\nu(s))$;
  $\nu(s) := G^{k-1} \circ D(\nu(s))$
fin pour

De (4.7) on déduit :

$$G^{n-1}(\nu(r)) = G^{n-1}(2^{n-1}) = 1.$$

On a ainsi $\nu(s) = 1$ à l'entrée de la boucle **pour**. Les assertions suivantes écrites entre accolades dans la boucle résultent alors aisément par récurrence sur j de (4.9) et (4.10) [en écrivant (4.9) sous la forme : $l = G^{-1} \circ D^{-v_2(l)+1}(l-1)$ où $l = \nu(s)+1$] :

pour j **de** 2 **à** $2^n - 2$ **pas** 2 **faire**
  $\{\nu(s) = j-1\}$
  $k := v_2(j)$;
  $\nu(s) := G^{-1} \circ D^{-k+1}(\nu(s))$;
  $\{\nu(s) = j\}$
  $\nu(s) := G^{k-1} \circ D(\nu(s))$
  $\{\nu(s) = j+1\}$
fin pour

Ces assertions montrent que $\nu(s)$ décrit les entiers de 1 à $2^n - 1$, ce que nous voulions montrer.

Revenons à l'algorithme général avec la forme équivalente suivante, plus commode pour l'application que nous allons en faire au problème des Tours de Hanoï :

$s := g^{n-1}(r)$;
pour j **de** 2 **à** $2^n - 1$ **faire**
  si j pair **alors**
    $k := v_2(j)$;
    $s := g^{-1} \circ d^{-k+1}(s)$
  sinon
    $s := g^{k-1} \circ d(s)$
  fin si
fin pour

[$v_2(j)$ n'est calculé qu'à une itération sur deux en commençant par la première; l'entier k reste donc le même à l'itération suivante.]

Cet algorithme général *permet une dérécursivation optimisée de tout programme récursif comportant deux appels récursifs, un au début et un à la fin, et tel que les fonctions donnant les arguments de ces deux appels sont inversibles.*

5. APPLICATION AU PROBLÈME DES TOURS DE HANOÏ

Les sommets de l'arborescence considérée pour le problème des Tours de Hanoï sont, comme on l'a vu, définis par les données des appels de la procédure « déplacer », à savoir les triplets de la forme (i, p, q) où $i \in \{1, 2, \ldots, n\}$ et $p, q \in \{1, 2, 3\}$ (rappelons que i représente le numéro du disque déplacé, p et q les piquets concernés par le déplacement).

On a ici $g(i, p, q) = (i-1, p, r)$ et $d(i, p, q) = (i-1, r, q)$.

Il est facile d'en déduire :

$$g^{n-1}(n, 1, 2) = (1, \ldots),$$
$$g^{-1} \circ d^{-k+1}(1, \ldots) = (k+1, \ldots),$$

et

$$g^{k-1} \circ d(k+1, \ldots) = (1, \ldots).$$

Si bien que dans l'application de l'algorithme général du paragraphe précédent, on peut obtenir, par récurrence sur j, une expression de la première composante du triplet; on aura en effet :

$$i = 1 \text{ au départ,}$$
$$i = k+1 = v_2(j)+1 \quad \text{dans le cas } j \text{ pair,}$$

et

$$i = 1 \text{ dans le cas } j \text{ impair.}$$

Remarque : Comme dans ce dernier cas on a $v_2(j) = 0$ on a en fait pour tout j :

$$i = v_2(j) + 1.$$

Cette expression sera utile plus loin.

On peut donc dans l'algorithme se restreindre à ne considérer que le couple (p, q). C'est ce que nous faisons dans ce qui suit, où on continue de noter g et d les fonctions restreintes aux couples (p, q) c'est-à-dire définies maintenant par : $g(p, q) = (p, r)$ et $d(p, q) = (r, q)$.

Ces fonctions étant involutives, $g^2 = d^2 = \mathrm{id}$, on a :

$$g^{-1} \circ d^{-k+1} = \textbf{si } k \text{ pair } \textbf{alors } g \circ d \textbf{ sinon } g \textbf{ fin si}$$

$$g^{k-1} \circ d = \textbf{si } k \text{ pair } \textbf{alors } g \circ d \textbf{ sinon } d \textbf{ fin si}.$$

L'algorithme précédent peut être réécrit dans le cas de notre problème, en insérant les actions d'affichage à effectuer à chaque sommet de l'arborescence :

```
i := 1;
(p, q) := g^(n-1) (1, 2);
afficher (i ' de ' p ' à ' q);
pour j de 2 à 2^n - 1 faire
  si j pair alors
    k := v2(j); i := k + 1;
    (p, q) := si k pair alors g ∘ d(p, q) sinon g(p, q) fin si
  sinon
    i := 1;
    (p, q) := si k pair alors g ∘ d(p, q) sinon d(p, q) fin si
  fin si;
  afficher (i ' de ' p ' à ' q)
fin pour
```

Et en explicitant les fonctions $g \circ d$, g et d on obtient le premier programme itératif simple donnant la solution du problème des Tours de Hanoï :

```
i := 1; p := 1;
si n pair alors q := 3 sinon q := 2 fin si;
afficher (i ' de ' p ' à ' q);
pour j de 2 à 2^n - 1 faire
  si j pair alors
    k := v2(j); i := k + 1;
    si k pair alors
      (p, q) := (6 - p - q, p)
    sinon
      q := 6 - p - q
    fin si
  sinon
    i := 1;
    si k pair alors (p, q) := (6 - p - q, p) sinon p := 6 - p - q fin si
  fin si;
  afficher (i ' de ' p ' à ' q)
fin pour
```

6. FORMULE DE RÉCURRENCE, DEUXIÈME PROGRAMME ITÉRATIF

Le programme précédent permet de démonter la formule de récurrence suivante, à ma connaissance inédite, et qui, comme on le verra, joue un rôle tout à fait fondamental dans ce problème.

$$t(j) = \varepsilon^{\mu(j)} \circ \delta \circ \varepsilon^{\mu(j-1)}(t(j-1)) \tag{6.1}$$

où : $t(j)$ désigne le couple de piquets concernés par le j-ième déplacement de disque

$\delta = g \circ d$: $(p, q) \to (r, p)$

ε: $(p, q) \to (q, p)$ (fonction d'inversion du couple).

$\mu(j)$ = reste de $v_2(j)$ modulo 2

Justification de (6.1) : Dans les différents cas qui peuvent se présenter on vérifie l'égalité de ce que donne le programme précédent et ce que donne la formule (6.1).

cas :	le programme donne :	(6.1) donne :
j pair, $k = v_2(j)$ pair	$g \circ d$	δ
j pair, $k = v_2(j)$ impair	g	$\varepsilon \circ \delta$
j impair, $k = v_2(j-1)$ pair	$g \circ d$	δ
j impair, $k = v_2(j-1)$ impair	d	$\delta \circ \varepsilon$

La formule (6.1) se prête à une interprétation algorithmique qui donne un deuxième programme itératif, extrêmement simple. On y voit le rôle fondamental joué dans la solution par l'application δ.

```
i := 1; p := 1;
si n pair alors q := 3 sinon q := 2 fin si;
afficher (i ' de ' p ' à ' q);
pour j de 2 à 2^n − 1 faire
  (p, q) := (6 − p − q, p);
  k := v2(j); i := k + 1;
  si k pair alors
    afficher (i ' de ' p ' à ' q)
  sinon
    afficher (i ' de ' q ' à ' p)
  fin si
fin pour
```

Ce programme se prête à une réalisation concrète en machine très efficace, en explicitant la détermination de $v_2(j)$, qui est, rappelons-le, l'exposant de 2 dans la décomposition en facteurs premiers de j, et qui est donc aussi ce

fameux nombre de 0 à partir de la droite dans la représentation binaire de l'entier j (qu'il est facile de déterminer en machine).

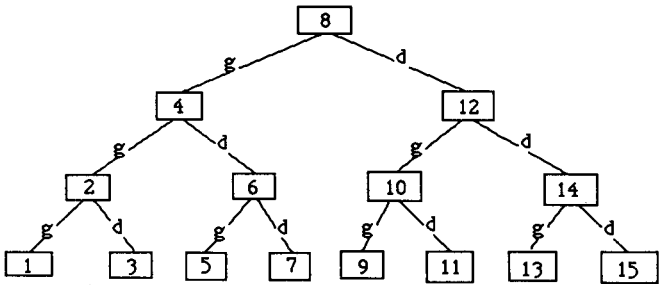


Figure 4

7. STRATÉGIES ITÉRATIVES

En remarquant que $\delta^3 = \mathrm{id}$, la formule (6.1) conduit à une stratégie itérative très simple.

On a en effet pour j de 4 à $2^n - 1$:

$$t(j) = \textbf{si } v_2(j) \equiv v_2(j-3) \bmod 2 \textbf{ alors } t(j-3) \textbf{ sinon } \varepsilon(t(j-3)) \textbf{ fin si} \quad (7.1)$$

Justification de (7.1) :

$$t(j) = \varepsilon^{\mu(j)} \circ \delta \circ \varepsilon^{\mu(j-1)} \circ \varepsilon^{\mu(j-1)} \circ \delta \circ \varepsilon^{\mu(j-2)} \circ \varepsilon^{\mu(j-2)} \circ \delta \circ \varepsilon^{\mu(j-3)}(t(j-3))$$
$$= \varepsilon^{\mu(j)} \circ \delta^3 \circ \varepsilon^{\mu(j-3)}(t(j-3)) = \varepsilon^{\mu(j)+\mu(j-3)}(t(j-3))$$

et on a

$$\mu(j) + \mu(j-3) \equiv 0 \bmod 2 \text{ si et seulement si } v_2(j) \equiv v_2(j-3) \bmod 2.$$

La formule (7.1) montre que la solution du problème des Tours de Hanoï est en fait *la répétition des* 3 *premiers déplacements, éventuellement inversés suivant la parité de* $v_2(j)$, *où* j *est le numéro du déplacement.* En effet δ reproduit la suite :

$$\begin{array}{ll} (1,3),\ (2,1),\ (3,2) & \text{si } n \text{ est pair,} \\ (1,2),\ (3,1),\ (2,3) & \text{si } n \text{ est impair.} \end{array}$$

Cette solution est si simple qu'on peut facilement en imaginer la réalisation par une « machine » même mécanique...

Comme autres conséquences des formules (6.1) et (7.1) on peut retrouver et parfois préciser les solutions itératives proposées dans la littérature. Ainsi la règle de Lavallée citée en introduction sur la suite de la somme des numéros des piquets concernés dans les déplacements, qui est périodique de période 3. C'est un corollaire immédiat de la formule (7.1). Mais ici on a plus : c'est la suite des *couples* de piquets concernés qui est déterminée par la formule (7.1), pas seulement la suite des *paires* (c'est-à-dire sans ordre), comme c'est le cas en considérant la somme des numéros des piquets (l'ordre est perdu). Ainsi on n'a plus à connaître l'état du jeu (à chaque coup on connaît le piquet de départ et le piquet d'arrivée du déplacement).

De (6.1), ou (7.1), on déduit facilement encore :

$$t(j) = t(j-6). \tag{7.2}$$

Rappelons qu'au j-ième coup c'est le disque de numéro $i = v_2(j) + 1$ qui est déplacé. Les coups impairs sont donc ceux qui déplacent le disque de numéro 1 c'est-à-dire le plus petit disque [si j est impair $v_2(j) = 0$ et donc $i = 1$]. De (7.2) il résulte facilement que la suite des coups impairs est elle aussi périodique de période 3. Ce qui se traduit par le fait souvent remarqué, sinon toujours justifié simplement, que le petit disque se déplace un coup sur deux et toujours dans le même sens. Cette propriété n'est en fait qu'un cas particulier d'une propriété générale qui se trouve dans Arsac [1] et plus explicitement dans Er [4], et que nous allons également retrouver très simplement grâce à la formule (6.1).

On déduit de (6.1) la formule :

$$t(j + 2^i) = \sigma(t(j)) \tag{7.3}$$

pour $j = 2^{i-1}$, $2^{i-1} + 2^i$, $2^{i-1} + 2.2^i$, ... et $i = 1, 2, \ldots, n$ où

$$\sigma : (p, q) \to (q, r).$$

Justification de (7.3) : On a

$$t(j + 2^i) = \varepsilon^{\mu(j+2^i)} \circ \delta^{2^i} \circ \varepsilon^{\mu(j)}(t(j))$$

pour

$$j = 2^{i-1}, 2^{i-1} + 2^i, 2^{i-1} + 2.2^i, \ldots$$

et

$$v_2(j+2^i)=v_2(j)=i-1, \qquad \mu(j+2^i)=\mu(j)=\begin{cases} 1 \text{ si } i \text{ est pair} \\ 0 \text{ si } i \text{ est impair} \end{cases}$$

car

$$2^i=\begin{cases} 1 \mod 3 \text{ si } i \text{ est pair} \\ 2 \mod 3 \text{ si } i \text{ est impair} \end{cases}$$

d'où

$$t(j+2^i)=\begin{cases} \varepsilon\circ\delta\circ\varepsilon(t(j)) \text{ si } i \text{ est pair} \\ \delta^2(t(j)) \text{ si } i \text{ est impair} \end{cases}$$

or

$$\varepsilon\circ\delta\circ\varepsilon=\delta^2=\sigma : (p, q)\to(q, r).$$

Comme le disque de numéro i est déplacé tous les 2^i coups, à partir du coup 2^{i-1} (considérer le niveau i de l'arborescence), il résulte de cette dernière formule que tous les disques de numéros impairs se déplacent dans le même sens, tous les disques de numéros pairs se déplacent dans l'autre sens. Plus précisément, chaque disque, de numéro pair ou impair, se meut suivant la fonction σ et c'est son premier déplacement qui détermine le sens cyclique de son mouvement qu'il conserve ensuite.

8. REMARQUE SUR LE CARACTÈRE ITÉRATIF DES SOLUTIONS

Les solutions dites itératives que nous venons d'évoquer, et qui ne nécessitent pas de connaître l'état du jeu, passe généralement par la quantité que nous avons notée $v_2(j)$, ou quelque chose d'équivalent, et qui est le nombre de 0 à droite de la représentation binaire du numéro du coup. En fait ce « compteur binaire » simule le fonctionnement d'une pile de dérécursivation qui permet le contrôle du parcours de l'arborescence (cf. *fig.* 5).

Cette observation peut amener à nourrir quelques doutes sur le caractère « purement itératif » avancé par les auteurs de ces solutions.

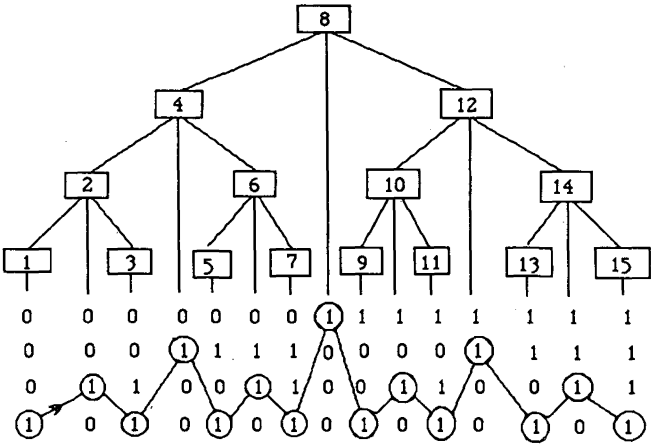


Figure 5

9. CONCLUSION

La récursivité est souvent considérée par les débutants comme un peu mystérieuse. En fait elle n'est jamais qu'une façon pour l'esprit humain d'écrire simplement, avec une grande économie de pensée, certains algorithmes. Elle ne peut être bien comprise que quand on a suivi pas à pas la dérécursivation, qu'il s'agit d'ailleurs ensuite d'optimiser. On s'aperçoit alors que la récursivité n'est rien d'autre qu'un moyen de définir, de façon *implicite*, une stratégie qu'il ne serait pas simple de trouver directement et que la dérécursivation *explicite*. Mais encore faut-il, pour bien comprendre, faire tout le chemin de l'implicite à l'explicite, comme nous venons de le faire ici pour le problème des Tours de Hanoï.

D'autres problèmes peuvent être ainsi traités et qui peuvent être beaucoup plus compliqués dans la mise en œuvre de la dérécursivation, notamment pour ce qui concerne l'inversion des fonctions transformant les données et aussi, aspect caché ici, pour la simplification permise par l'associativité d'une loi de composition apparaissant dans une définition récursive.

REMERCIEMENTS

L'auteur tient à remercier Raymond Durand pour une conversation utile sur le caractère récursif ou itératif du problème des Tours de Hanoï, et un rapporteur qui par ses indications a permis à l'auteur la mise au point de la justification de l'algorithme général du paragraphe 4.

BIBLIOGRAPHIE

1. J. ARSAC, *La construction des programmes structurés*, Dunod Informatique, 1977.
2. J. ARSAC, *Les bases de la programmation structurée*, Dunod Informatique, 1983.
3. P. BUNEMAN et L. LEVY, *The Towers of Hanoï Problem*, Information Processing letters, vol. 10, 1980, p. 243-244.
4. M. C. ER, *A Representation Approach to the Tower of Hanoï Problem*, The Computer Journal, vol. 25, 1982, p. 442-447.
5. J. HARDOUIN DUPARC, *Génération de mots à définition récursive par des piles d'automates*, preprint, 1985.
6. P. J. HAYES, *Discussion and Correspondence. A Note on the Towers of Hanoï Problem*, The Computer Journal, 1977, p. 282-285.
7. I. LAVALLÉE, *Note sur le problème des Tours de Hanoï*, Rev. Roumaine Math. pures appl., vol. 30, 1985, p. 433-438.
8. B. MEYER et C. BAUDOIN, *Méthodes de programmation*, 3e éd., Eyrolles, 1984.