(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
4 septembre 2003 (04.09.2003)

PCT

(10) Numéro de publication internationale
WO 03/073320 A2

(51) Classification internationale des brevets[7] : G06F 17/30

(21) Numéro de la demande internationale :
PCT/FR03/00576

(22) Date de dépôt international :
21 février 2003 (21.02.2003)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
02/02664 27 février 2002 (27.02.2002) FR

(71) Déposant *(pour tous les États désignés sauf US)* : **FRANCE TELECOM SA** [FR/FR]; 6, place d'Alleray, F-75015 Paris (FR).

(72) Inventeur; et
(75) Inventeur/Déposant *(pour US seulement)* : **LASSALLE, Edmond** [FR/FR]; 17, Rue du Hingar, F-22300 Lannion (FR).

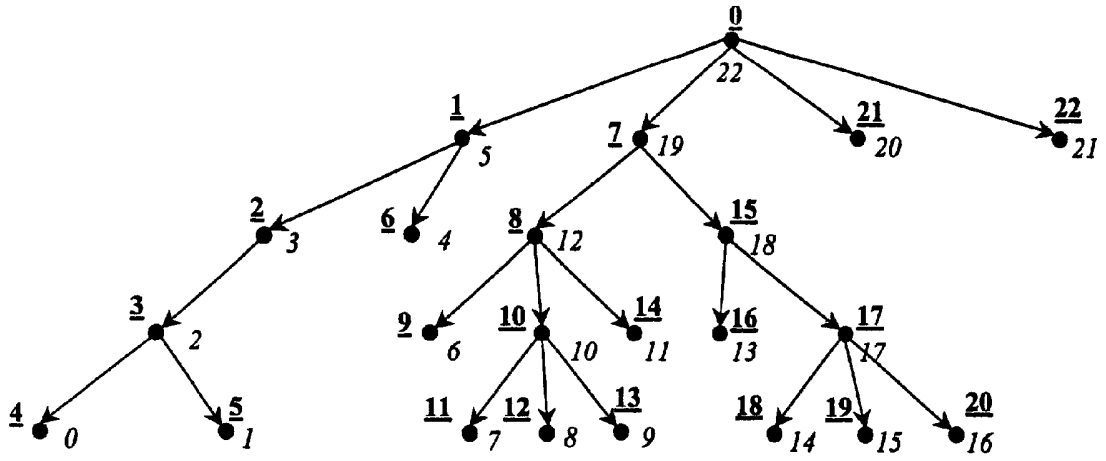
(74) Mandataire : **MAILLET, ALAIN**; Cabinet Le Guen & Maillet, 5, place Newquay - BP 70250, F-35802 Dinard Cedex (FR).

(81) États désignés *(national)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ,

[Suite sur la page suivante]

(54) Title: COMPUTER REPRESENTATION OF A DATA TREE STRUCTURE AND THE ASSOCIATED ENCODING/DECODING METHODS

(54) Titre : REPRÉSENTATION INFORMATIQUE D'UNE STRUCTURE DE DONNÉES ARBORESCENTE ET MÉTHODES DE CODAGE/DÉCODAGE ASSOCIÉES



(57) Abstract: The invention relates to a computer representation of a directed tree that is representative of the organisation of a data set, such as a data dictionary, each piece of data being associated with a node of the tree. The aforementioned representation comprises a table of values which are stored in a memory unit, said values being representative of the rows of nodes of said tree which are arranged according to a first total order relation. Moreover, the respective addresses at which said values are stored are representative of the rows of nodes of the tree which are arranged according to a second total order relation. The invention also relates to a method of encoding the directed tree and to the encoding of a datum from said set and the decoding of an index that is representative of a datum from said set.

(57) Abrégé : L'invention concerne une représentation informatique d'un arbre orienté représentatif de l'organisation d'un ensemble de données, notamment d'un dictionnaire de données, chaque donnée étant associée à un nœud de l'arbre. Cette représentation comprend une table de valeurs stockées dans une mémoire, lesdites valeurs étant représentatives des rangs des nœuds dudit arbre ordonnés selon une première relation d'ordre total, les adresses respectives auxquelles

[Suite sur la page suivante]

WO 03/073320 A2

DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) États désignés *(régional)* **:** brevet ARIPO (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), brevet eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), brevet européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, SE, SI, SK, TR), brevet OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :

— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

lesdites valeurs sont stockées étant représentatives des rangs des nœuds dudit arbre ordonnés selon une seconde relation d'ordre total. L'invention concerne également une méthode de codage dudit arbre orienté ainsi que le codage d'une donnée dudit ensemble et le décodage d'un index représentatif d'une donnée dudit ensemble.

Représentation informatique d'une structure de données arborescente et méthodes de codage/décodage associées

La présente invention concerne une représentation informatique d'un arbre orienté représentant l'organisation d'un ensemble de données, notamment d'un dictionnaire. La présente invention concerne également une méthode de codage dudit arbre orienté en ladite représentation informatique. La présente invention concerne encore une méthode de codage d'une donnée appartenant audit ensemble en un index de ladite représentation informatique. La présente invention concerne enfin une méthode de décodage permettant de retrouver à partir d'un index de ladite représentation informatique ladite donnée correspondante.

Préalablement à l'exposé de l'état de la technique, il convient de définir un certain nombre de termes qui seront utilisés par la suite.

On appelle graphe orienté (dénommé ci-après simplement graphe) un couple G=(S,A) où S est un ensemble de sommets (dénommés ci-après également nœuds) et A est un sous-ensemble de SxS, appelé ensemble des arcs.

Un chemin du graphe est une suite ordonnée $(s_0,s_1,..,s_n)$ de sommets tels que (s_{i-1},s_i), est un arc, pour i=1,..n. Lorsque $s_n=s_0$ avec $n\geq 1$, le chemin est appelé circuit ou

cycle. Un graphe est dit connexe si deux nœuds quelconques de ce graphe sont reliés par un chemin.

Un arbre est défini comme un graphe connexe sans circuit. On peut montrer que deux sommets quelconques d'un arbre sont reliés par un chemin unique. Un arbre possède un sommet particulier R, tel que tout sommet s distinct de R est relié à ce dernier par un chemin. Ce sommet particulier est appelé racine de l'arbre.

Pour un sommet S donné, on appelle descendant de S tout sommet s_d de l'arbre tel qu'il existe un chemin entre s et s_d. Réciproquement, pour un sommet s donné, on appelle ancêtre de s tout sommet S_a de l'arbre tel qu'il existe un chemin entre S_a et s. On appelle sommet fils d'un sommet S un descendant s_f de s tel que $(S, s_f) \in A$. Pour tout sommet s, de l'arbre on appelle sous-arbre issu de s, l'arbre de racine s comprenant tous les descendants de s.

Enfin , on appelle feuille tout sommet de l'arbre ne possédant pas de descendant.

De nombreuses méthodes de traitement de l'information font appel à une représentation des données selon une structure arborescente (ou arbre), notamment des méthodes de classification, de compression ou de stockage d'information.

Selon le type d'application considéré, les données peuvent être des chaînes de caractères, des suites de phonèmes, des formes d'onde, des patterns de luminance/ chrominance etc.

Sans perte de généralité, nous considérerons par la suite que les données sont constituées de chaînes d'entités élémentaires ou caractères (par exemple des lettres, des idéogrammes, des chiffres, des signes alpha-numériques). L'ensemble de ces caractères possibles constitue un alphabet. Nous supposerons que cet alphabet est muni d'une relation d'ordre total, appelé ordre alphabétique.

Dans de nombreuses applications de types « moteurs de recherche », « recherche dans un annuaire », « recherche dans un dictionnaire » etc., un volume très important de données doit pouvoir être stocké et accédé, ce qui impose des contraintes sévères en conditions d'exploitation réelle, notamment pour des accès en ligne.

Chaque donnée doit pouvoir être accessible rapidement sans faire appel à de grandes ressources de calcul. Aussi, pour réduire les temps d'accès, des volumes importants de données doivent résider en mémoire centrale. Afin de ne pas augmenter démesurément la taille de cette mémoire, il est souvent nécessaire d'effectuer une compression préalable des données. Avantageusement, les données doivent pouvoir

être accédées sans avoir à être décompressées, ce qui pénaliserait là encore les temps d'accès.

Dans les types d'applications considérées plus haut, un traitement dissymétrique des données est envisageable : la phase de compression des données peut comprendre un traitement relativement long et complexe alors que celle permettant leur accès et leur récupération doit être simple et rapide. Les données peuvent être ainsi stockées en mémoire sous une forme compressée et figée, leur mise à jour étant réalisée « off-line» avant qu'elles soient rebasculées en ligne.

Il existe une structure d'organisation des données qui se prête particulièrement bien à la compression : celle d'arbre telle que définie plus haut. Elle se rencontre notamment dans les dictionnaires ou les annuaires. Un dictionnaire au sens commun du mot est un fichier de données (appelée également entrées), chaque donnée étant constituée d'une chaîne de caractères alphabétiques, ces chaînes étant organisées selon une stucture arborescente.

En pratique, dans une représentation informatique, chaque donnée du dictionnaire est associé à un index. La recherche d'une chaîne de caractères (ou mot) dans le dictionnaire revient à identifier l'index du mot correspondant. Ainsi un texte peut être représenté par une suite d'index, plus adaptée au traitement informatique que la représentation initiale.

Différents types de représentation ont été proposés dans l'état de la technique et notamment :

- sous forme de table de dichotomie, avec utilisation d'une compression de Ziv-Lempel
- sous forme de table de hash
- sous forme d'arbre lexical

Ces différents types de représentation conduisent à des performances équivalentes en cas d'accès parfait. On appelle accès parfait un mode d'accès où l'on cherche la chaîne de caractères exacte dans le dictionnaire correspondant au mot à analyser, sans prendre en compte les erreurs ou les altérations.

La représentation par arbre lexical suppose une analyse (ou « parsing ») des chaînes de caractères. Un exemple d'arbre lexical est donné en Fig. 1 pour le dictionnaire Δ suivant :

Δ = {abolish, abolition, appeal, attorney, bar, barrister, bench, case, court, crime}

On remarque que dans l'arbre lexical, les arcs sont associés aux caractères des mots du dictionnaire. Plus précisément, à chaque arc de l'arbre est associée une étiquette, chaque étiquette ayant comme label un caractère. L'arbre lexical est la réunion de tous les chemins dont le squelette correspond à un mot du dictionnaire. On appelle squelette d'un chemin la chaîne des caractères des étiquettes des arcs constituant ce chemin. Un mot du dictionnaire est encore appelé « entrée » du dictionnaire.

On notera que les feuilles de l'arbre lexical ont été représentées par des cercles alors que les autres sommets ont été représentés par des disques. La racine de l'arbre a été indiquée par *R*.

On suppose que l'arbre est indexé, c'est-à-dire qu'à chaque sommet est associé un index. Une opération élémentaire sur un arbre lexical est de rechercher à partir d'un mot donné, l'index de l'entrée du dictionnaire correspondante. Cette opération nécessite de parcourir l'arbre selon les arcs étiquetés par les caractères successifs composant le mot.

Plus particulièrement, l'algorithme de recherche met en œuvre une fonction *AnalyserMot* qui retourne comme valeur l'index (index-associé (*s*)) de l'entrée du dictionnaire si cette dernière y est présente, ou, à défaut, un code de non-identification (Index-Mot-Inconnu). Elle est donnée ci-après en pseudo-code C :

```
Fonction NumeroIndex AnalyserMot (chaine MotAnalyse, sommet Racine)
Debut
        sommet s =Racine ;
        Pour chaque Caractère de MotAnalyse Faire
                Si fin-de-mot (MotAnalyse) et s est une feuille
                        Alors retourner index-associé (s) ;
                Si Caractère correspond à une étiquette d'un arc issu de s
                        Alors s=descendant-correspondant(s) ;
```

Sinon retourner Index-Mot-Inconnu ;

Fin

La navigation par les instructions s=descendant-correspondant(s) suppose que l'on dispose d'une représentation informatique de l'arbre lexical.

De manière générale, il est nécessaire de disposer d'une représentation informatique d'un arbre pour pouvoir le parcourir, l'utiliser et le modifier aisément.

Selon, une première représentation informatique connue, un arbre est représenté par un tableau d'adjacence $M=(m_{ij})$ $i=0,..,n$; $j=0,..,n$ stocké en mémoire avec $m_{ij}=1$ si $(s_i,s_j)\in A$.

Selon une représentation informatique plus courante, un arbre est représenté comme une séquence de pointeurs informatiques. Selon une première variante connue illustrée en Fig. 2A, chaque nœud est représenté par une valeur (ou index) et un tableau de pointeurs pointant vers ses nœuds fils. La taille du tableau correspond au nombre maximum de fils (k) que peut posséder un nœud de l'arbre (l'arbre est dit en ce cas « k-aire »). La Fig. 2A donne un exemple de représentation d'un arbre 3-aire selon cette variante.

Le codage des fils d'un nœud par un tableau de pointeurs présente l'inconvénient de consommer beaucoup d'espace mémoire lorsque l'arbre contient un petit nombre de nœuds ayant beaucoup de fils et de nombreux autres nœuds ayant peu de fils. Selon une seconde variante connue de représentation, on remédie à cette difficulté en utilisant pour un nœud donné un pointeur vers l'un de ses nœuds fils, appelé fils aîné, et un pointeur du fils aîné vers une liste chaînée de ses frères. La Fig. 2B donne un exemple de représentation selon cette seconde variante, pour un arbre 5-aire.

Les pointeurs permettent de modifier rapidement la structure de l'arbre mais ils sont relativement coûteux en espace mémoire. Qui plus est, la détection d'une relation de descendance entre de nœuds n'est pas immédiate. Elle suppose de déterminer le chemin qui relie les deux nœuds, ce qui, dans le cadre d'une représentation par pointeurs, impliquent des ressources de calcul importantes. On peut réduire considérablement la quantité de calculs à effectuer en mémorisant la fermeture transitive de l'arbre, auquel cas un nœud pointe vers chacun de ses descendants. Cependant, cette dernière option est particulièrement gourmande en espace mémoire.

Le problème à la base de l'invention est de proposer une représentation informatique d'un arbre qui n'occupe que peu d'espace mémoire, qui permette de le parcourir aisément et de le modifier simplement.

Ce problème est résolu par la représentation informatique d'un arbre représentatif de l'organisation d'un ensemble de données, en particulier d'un dictionnaire de données, chaque donnée étant associée à un nœud particulier dudit arbre, cette représentation comprenant une table de valeurs stockées dans une mémoire, lesdites valeurs étant représentatives des rangs des nœuds dudit arbre ordonnés selon une première relation d'ordre total, les adresses auxqelles lesdites valeurs étant stockées étant représentatives des rangs des nœuds dudit arbre ordonnés selon une seconde relation d'ordre total.

Avantageusement, la première relation d'ordre total est une combinaison d'une relation d'ordre de descendance ordonnant un nœud par rapport à ses descendants et d'une relation d'ordre de primogéniture ordonnant les nœuds fils d'un même nœud.

Selon un premier mode de réalisation, un premier nœud de l'arbre est inférieur à un second nœud de l'arbre selon ladite première relation d'ordre total si le second nœud est un descendant du premier nœud ou si, l'ancêtre commun au premier et au second nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon la relation d'ordre de primogéniture.

Selon un second mode de réalisation, un premier nœud de l'arbre est supérieur à un second nœud de l'arbre selon ladite première relation d'ordre total si le second nœud est un descendant du premier nœud ou si, l'ancêtre commun au premier et au second nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon la relation d'ordre de primogéniture.

Avantageusement, la seconde relation d'ordre total est une combinaison de la relation d'ordre inverse de ladite relation d'ordre de descendance et de ladite relation d'ordre de primogéniture.

Selon une première variante, un premier nœud de l'arbre est inférieur à un second nœud de l'arbre selon ladite seconde relation d'ordre total si le premier nœud est un descendant du second nœud ou si, l'ancêtre commun au premier et au second

nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon ladite relation d'ordre de primogéniture.

Selon une seconde variante, un premier nœud de l'arbre est supérieur à un second nœud de l'arbre selon ladite seconde relation d'ordre total si le premier nœud est un descendant du second nœud ou si, l'ancêtre commun au premier et au second nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon ladite relation d'ordre de primogéniture .

Si les données sont des suites de caractères d'un alphabet muni d'un ordre alphabétique, chaque arc dudit arbre étant associé à un caractère d'au moins une donnée, la relation d'ordre de primogéniture entre deux fils d'un même nœud peut être donnée par la relation d'ordre alphabétique entre les caractères associés aux arcs respectifs entre ledit nœud et ses deux fils.

L'invention concerne également une méthode de codage d'un arbre orienté représentatif de l'organisation d'un ensemble de données, notamment d'un dictionnaire, chaque donnée dudit ensemble étant associée à un nœud particulier dudit arbre, dans laquelle on attribue à chaque nœud dudit arbre un premier et un second index, le premier index étant représentatif du rang du nœud selon une première relation d'ordre total ordonnant les nœuds dudit arbre, le second index étant représentatif du rang du nœud selon une seconde relation d'ordre total, la première relation d'ordre total étant une combinaison d'une relation d'un ordre de descendance ordonnant un nœud par rapport à ses descendants et d'une relation d'ordre de primogéniture ordonnant les nœuds fils d'un même nœud, la seconde relation d'ordre total étant une combinaison de la relation d'ordre inverse de ladite relation d'ordre de descendance et de ladite relation d'ordre de primogéniture.

Avantageusement la méthode de codage comprend un appel récursif d'une étape de calcul fournissant pour un nœud quelconque de l'arbre, la taille du sous-arbre issu dudit nœud.

Pour un premier fils et un second fils d'un même nœud, dit nœud père, adjacents dans une liste des fils ordonnée selon ladite relation l'ordre de primogéniture, l'étape de calcul détermine le premier index du second fils à partir du

premier index du premier fils et de la taille du sous-arbre issu du premier fils, et le second index du second fils à partir du second index du premier fils et de la taille du sous-arbre issu du second fils.

Ladite étape de calcul détermine le premier index du fils classé premier dans ladite liste à partir du premier index dudit nœud père et le second index dudit nœud père à partir du second index du fils classé dernier dans ladite liste.

Ladite étape de calcul détermine également la taille du sous-arbre issu dudit nœud père à partir de la somme des tailles des sous-arbres issus de ses fils.

Avantageusement, ladite méthode de codage opère sur une première représentation dudit arbre au moyen de pointeurs dans laquelle, pour un nœud donné, un premier type de pointeur fournit un nœud fils selon la relation d'ordre de descendance et un second type de pointeur fournit la liste de ses autres fils.

L'invention est également définie par une méthode de codage d'une donnée d'entrée appartenant à un ensemble de données organisées selon une structure d'arbre orienté, notamment à un dictionnaire de données, les données étant formées de suites de caractères d'un alphabet muni d'un ordre alphabétique, chaque donnée étant associée à un nœud particulier dudit arbre et à chaque arc dudit arbre étant associé un caractère, dans laquelle ledit arbre étant représenté au moyen de la représentation informatique précité, on parcourt l'arbre de nœud en nœud selon un chemin partant de la racine et l'on analyse ladite donnée d'entrée caractère par caractère, le nœud suivant d'un nœud courant dudit chemin étant choisi parmi les fils de ce dernier, le choix étant effectué au moyen d'une succession d'étapes de comparaison, chaque étape de comparaison comparant le caractère en cours de ladite donnée d'entrée et le caractère associé à l'arc reliant le nœud courant à l'un de ses fils, le parcours n'étant interrompu que lorsque la ladite donnée d'entrée est entièrement analysée, la méthode fournissant comme valeur codée de ladite donnée d'entrée un index fonction de l'adresse de la table de ladite représentation informatique représentative du dernier nœud dudit chemin .

L'invention est encore définie par une méthode de décodage d'un index représentatif d'une donnée appartenant à un ensemble de données organisées selon une structure d'arbre orienté, en particulier d'un dictionnaire de données, les données étant formées de suites de caractères d'un alphabet muni d'un ordre alphabétique, chaque donnée étant associée à un nœud particulier dudit arbre et à chaque arc dudit arbre étant associé un caractère, dans laquelle ledit arbre étant représenté au moyen

de la représentation informatique précité, on parcourt l'arbre selon un chemin partant de la racine, le nœud suivant un nœud courant dudit chemin étant choisi parmi les fils de ce dernier, le choix étant effectué au moyen d'une succession d'étapes de comparaison, chaque étape de comparaison comparant ledit index à un index représentatif de l'un desdits fils dans ladite représentation informatique, la méthode fournissant comme donnée décodée la chaîne des caractères associés aux arcs formant ledit chemin.

Les caractéristiques de l'invention mentionnées ci-dessus, ainsi que d'autres, apparaîtront plus clairement à la lecture de la description suivante de certains modes de réalisation, ladite description étant faite en relation avec les dessins joints, parmi lesquels :

La Fig. 1 représente un exemple d'arbre lexical ;

La Fig. 2A montre une première représentation informatique d'un arbre à l'aide de pointeurs ;

La Fig. 2B montre une seconde représentation informatique d'un arbre à l'aide de pointeurs ;

La Fig. 3A illustre sur un exemple une méthode de codage d'arbre selon un premier mode de réalisation de l'invention ;

La Fig. 3B montre une première variante de représentation informatique de l'arbre de la Fig. 3A ;

La Fig. 3C montre une seconde variante de représentation informatique de l'arbre de la Fig. 3A ;

La Fig. 4 illustre une portion d'arbre avant indexation par rang préfixe et rang postfixe.

L'idée à la base de l'invention est de créer une nouvelle représentation informatique d'un arbre à partir d'une relation d'ordre total traduisant les relations de dépendance entre nœuds.

La relation de dépendance entre nœuds induit une relation d'ordre partiel sur l'ensemble des nœuds de l'arbre. Par exemple, si l'on convient pour deux nœuds s_1 et s_2 de l'arbre que : $s_1 > s_2$ si et seulement si s_2 est un descendant de s_1, on dispose d'une ralation d'ordre. Cependant cet ordre n'est qu'un ordre partiel puisque tous les nœuds de l'arbre ne peuvent être ainsi comparés (par exemple les fils d'un même nœud).

On peut construire une relation d'ordre total sur les nœuds d'un arbre si l'on sait ordonner tous les fils d'un même nœud. L'ordre selon lequel on ordonne les fils d'un

même nœud sera conventionnellement appelé ordre de primogéniture. Pour un arbre lexical, dont les étiquettes des arcs contiennent des caractères alphabétiques, on peut convenir que deux fils s_1 et s_2 d'un même nœud S satisfont à la relation $s_1 > s_2$ si le caractère de l'étiquette associée à l'arc (S,s_1) précède celui de l'étiquette associée à l'arc (S,s_2). Autrement l'ordre alphabétique des labels des étiquettes induit un ordre de primogéniture sur les nœuds fils d'un même nœud .

La combinaison de la relation d'ordre partiel de descendance (notée ci-après $\underset{D}{>}$ ou $\underset{D}{<}$) et de la relation l'ordre de primogéniture (notée ci-après $\underset{P}{>}$ ou $\underset{P}{<}$) permet d'obtenir une relation d'ordre total sur l'ensemble des nœuds. Cette combinaison peut être réalisée de plusieurs manières différentes :

- relation d'ordre préfixe (notée conventionnellement $\underset{pref}{<}$) :

$$a \underset{pref}{<} b \text{ si } a \underset{D}{<} b \text{ ou } a' \underset{P}{<} b'$$

où a' et b' sont les fils de l'ancêtre commun à a et b, tels que a descend ou est confondu avec a' et b descend ou est confondu avec b'.

Autrement dit, le nœud a est inférieur au nœud b au sens de l'ordre préfixe si b est un descendant de a ou a' est un frère aîné de b'.

- relation d'ordre préfixe inverse (notée conventionnellement $\underset{frep}{<}$)

$$b \underset{frep}{<} a \text{ si } a \underset{D}{<} b \text{ ou } a' \underset{P}{<} b'$$

- relation d'ordre postfixe inverse (notée conventionnellement $\underset{post}{<}$) :

$$a \underset{post}{<} b \text{ si } a \underset{D}{>} b \text{ ou } a' \underset{P}{<} b'$$

autrement dit le nœud a est inférieur au nœud b au sens de l'ordre postfixe si a est un descendant de b ou a' est un frère aîné de b'.

- relation d'ordre postfixe inverse (notée conventionnellement $\underset{tsop}{<}$) :

$b <_{tsop} a$ si $a >_{D} b$ ou $a' <_{P} b'$

Etant donné que deux nœuds quelconques d'un arbre sont, soit descendants l'un de l'autre, soit descendants d'un ancêtre commun, les relations d'ordre définies ci-dessus sont des relations d'ordre total.

Les relations d'ordre $<_{pref}$ (ou $\leq_{frep}$) et $<_{post}$ (ou $\leq_{tsop}$) permettent donc d'ordonner entièrement l'ensemble S des nœuds de l'arbre. Autrement dit, à chacune de ces relations d'ordre on peut associer une fonction « rang » de S dans [0,n], par exemple :

RangPrefixe : S→[0,n]
telle que $s_1 <_{pref} s_2$ si et seulement si *RangPrefixe*(s_1)< *RangPrefixe*(s_2)

RangPostfixe : S→[0,n]
telle que $s_1 <_{post} s_2$ si et seulement si *RangPostfixe*(s_1)< *RangPostfixe*(s_2)

RangPrefixe et *RangPostfixe* sont des morphismes d'ensembles ordonnés. De même manière on peut définir des fonctions rang *RangPrefixeInverse* et *RangPostfixeInverse* à partir des relations d'ordre $\leq_{frep}$ et $<_{tsop}$:

RangPrefixeInverse : S→[0,n]
telle que $s_1 <_{frep} s_2$ ssi *RangPrefixeInverse*(s_1)< *RangPrefixeInverse*(s_2)

RangPostfixeInverse : S→[0,n]
telle que $s_1 <_{tsop} s_2$ ssi si *RangPostfixeInverse*(s_1)< *RangPostfixeInverse*(s_2)

Selon un premier mode de réalisation, on utilise, pour contruire la représentation informatique, une bijection T de [0,n] dans [0,n] définie par :

T : [0,n]→[0,n]
T=*RangPostfixe* o *RangPrefixe*$^{-1}$

Selon une variante de ce premier mode de réalisation, on utilise la bijection T^{-1}.

De la même façon, on pourra utiliser les bijections formées par composition : *RangPostfixeInverse* o *RangPrefixe*$^{-1}$, *RangPostfixe* o *RangPrefixeInverse*$^{-1}$ ou *RangPostfixeInverse* o *RangPrefixeInverse*$^{-1}$ dans d'autres modes de réalisation de l'invention ou bien encore, dans des variantes de ceux-ci, les inverses de ces bijections.

Dans un but de simplification, nous limiterons l'exposé de l'invention à l'utilisation des bijections T et T^{-1}, bien que les autres bijections puissent être également utilisées .

Ladite bijection T peut être représentée de manière informatique sous la forme d'un premier tableau de valeurs stockées en mémoire, le rang postfixe d'un nœud étant stocké à une adresse représentative du rang préfixe de ce nœud.

De même, la bijection T^{-1} peut être représentée de manière informatique sous la forme d'un second tableau de valeurs stockées en mémoire, le rang postfixe d'un nœud étant stocké à une adresse représentative du rang préfixe de ce nœud.

Un exemple fera mieux comprendre l'intérêt et l'utilisation de ces bijections.

La Fig. 3A illustre un arbre dont les nœuds ont été indexés par les rangs préfixe (en gras et souligné) et les rangs postfixe (en italique). La relation d'ordre de primogéniture, par exemple induite par un ordre alphabétique sur les labels des étiquettes dans le cas d'un arbre lexical, a été représentée conventionnellement dans le sens croissant de la gauche vers la droite. A chaque nœud s est ainsi associé un couple :

(*RangPrefixe*(s), *RangPostfixe*(s))

Ces couples sont avantageusement stockés dans un tableau au moyen de la bijection T (Fig. 3B) ou T^{-1} (Fig. 3C). En Fig. 3B les valeurs de rang postfixe ont été stockées aux adresses données par les valeurs de rang préfixe correspondantes. A l'inverse, en Fig. 3C les valeurs de rang prefixe ont été stockées aux adresses données par les valeurs de rang postfixe correspondantes.

Un premier avantage de la représentation informatique de l'arbre selon l'invention est de n'occuper qu'un espace mémoire de la taille de l'arbre (n+1) par opposition à une représentation classique par pointeurs (Figs. 2A et 2B) nécessitant une occupation mémoire au moins double.

Un second avantage essentiel de cette représentation informatique est de permettre très simplement de détecter une relation de dépendance entre deux nœuds de l'arbre. En effet, pour déterminer si un nœud s_2 est dépendant d'un nœud s_1, il suffit de comparer *RangPrefixe*(s_1) à *RangPrefixe*(s_2) d'une part, et *RangPostfixe*(s_1) à *RangPostfixe*(s_2), d'autre part :

s_2 est dépendant de s_1 si et seulement si l'on a:

RangPrefixe(s_2) > *RangPrefixe*(s_1) et *RangPostfixe*(s_2)< *RangPostfixe*(s_1)

Ainsi, dans l'exemple de la Fig. 3A, on vérifie que le nœud représenté par le couple (*RangPrefixe*, *RangPostfixe*)= (**5**,*1*) est bien dépendant de celui représenté par le couple (*RangPrefixe*, *RangPostfixe*)= (**1**,*5*) mais non de celui représenté par le couple (*RangPrefixe*, *RangPostfixe*)= (*22*, **21**).

De la même façon, grâce au tableau de la Fig. 3B, il est très facile de déterminer les descendants ou les ancêtres d'un nœud donné. Par exemple, pour déterminer la liste des descendants du nœud (**8**, *12*), il suffit de balayer le tableau dans le sens des adresses croissantes à partir de l'adresse **8** et de rechercher parmi les données stockées celles qui sont inférieures au postfixe *12* (ici *6*, *10*, *11*, *7*, *8*, *9*) . Ces valeurs indiquent les rangs postfixe des descendants du nœud en question. Pour déterminer la liste des ancêtres du nœud (**8**, *12*), il suffit de balayer le tableau dans le sens des adresses décroissantes à partir de l'adresse **8** et de rechercher parmi les données stockées celles qui sont supérieures au postfixe *12* (ici *19*, *22*). Ces valeurs indiquent les rangs postfixe des ancêtres du nœud en question.

On procède de manière duale à partir du tableau de la Fig. 3C. En reprenant l'exemple précédent, il suffit, pour déterminer la liste des descendants, de balayer le tableau dans le sens des adresses décroissantes à partir de l'adresse *12* et de rechercher parmi les données stockées celles qui sont supérieures au préfixe **8** (ici **14**, **10** ,**13** , **12**,

11, **9**). Ces valeurs indiquent les rangs préfixe des descendants du nœud en question. De même, pour déterminer la liste des ancêtres de (**8**, *12*), il suffit de balayer le tableau dans le sens des adresses croissantes à partir de l'adresse *12* et de rechercher parmi les données stockées celles qui sont inférieures au préfixe **8** (ici **7**, **0**).

Un troisième avantage de la représentation informatique selon l'invention est de permettre un parcours aisé de l'arbre, soit de la racine vers les feuilles, parcours que l'on effectue par exemple lorsque l'on analyse une chaîne de caractères (mot) à l'aide d'un arbre lexical, soit des feuilles vers la racine, parcours que l'on effectue par exemple lorsque l'on génère une chaîne de caractères à partir de l'index d'un nœud.

Le parcours de l'arbre de la racine vers les feuilles suppose que l'on sache déterminer les fils d'un nœud donné. Comme nous allons le voir, le tableau de la Fig. 3B (ou celui de la Fig. 3C) permet de les retrouver rapidement.

Supposons que l'algorithme de navigation recherche les fils du nœud (*12*, **8**) et considérons le tableau de la Fig. 3B. Le tableau est balayé dans le sens des adresses croissantes à partir de l'adresse **8** . Comme précédemment, on recherche les données du tabeau inférieures à *12*. Lorsque l'on retient une donnée *x* inférieure à *12*, on ne considère plus les données suivantes qui sont inférieures à *x*. Autrement dit, on continue à balayer le tableau jusqu'à ce que l'on trouve à nouveau une donnée *x'* supérieure à *x* (mais toujours inférieure à la valeur initiale *12*). On itère le processus jusqu'à la fin du tableau. Ainsi dans l'exemple présent, on rencontre tout d'abord la valeur *6* qui est retenue (*<12*) puis la valeur *10* qui est retenue également (*6<10<12*). Les valeurs suivantes *7*, *8*, *9* ne sont pas retenues car, si elles sont bien inférieures à *12*, elles ne sont pas supérieures à la dernière valeur retenue *10*. La valeur *11* est ensuite retenue (*10<11<12*) mais les valeurs suivantes ne peuvent l'être car supérieures à *12*.

On procède de manière duale à partir du tableau de la Fig. 3C. En reprenant l'exemple précédent, on balaye le tableau dans le sens des adresses décroissantes à partir de l'adresse *12* . Comme précédemment, on recherche les données stockées supérieures au préfixe **8**. Lorsque l'on rencontre une donnée **x** supérieure à **8** , on ne considère plus les données suivantes qui sont supérieures à **x**. Autrement dit, on continue à balayer le tableau jusqu'à ce que l'on trouve à nouveau une donnée **y** inférieure à **x** (mais toujours supérieure à la valeur initiale **8**). On itère le processus jusqu'à ce qu'on atteigne le début du tableau. Ainsi dans l'exemple présent, on

rencontre tout d'abord la valeur **14** qui est retenue (>**8**) puis la valeur **10** qui est retenue également (**8**<**10**<**14**). Les valeurs suivantes **11**, **12**, **13** ne sont pas retenues car, si elles sont bien supérieures à **8**, elles ne sont pas inférieures à la dernière valeur retenue **10**. La valeur **9** est ensuite retenue (**8**<**9**<**10**) mais les valeurs suivantes ne peuvent l'être car inférieures à **8**.

Cette manière de déterminer les fils d'un nœud donné peut convenir pour les arbres de faible taille. Cependant, comme on le verra plus loin, pour des arbres de grande taille, il est plus rapide de calculer directement les rangs prefixe/ postfixe desdits fils.

On notera que, si l'on avait employé, au lieu des tableaux construits à partir des bijections T et T^{-1}, des tableaux construits respectivement à partir des bijections *RangPostfixeInverse* o *RangPrefixe*$^{-1}$, *RangPostfixe* o *RangPrefixeInverse*$^{-1}$ ou *RangPostfixeInverse* o *RangPrefixeInverse*$^{-1}$ ou bien encore à partir des inverses de ces bijections, on aurait pu déterminer de manière similaire les fils d'un nœud donné, au prix éventuel d'un changement de sens de balayage et/ou de changement de sens des inégalités.

De la même façon, le parcours d'un arbre de ses feuilles vers sa racine suppose que l'on sache déterminer le père d'un nœud donné. Supposons que l'algorithme de navigation recherche le père du nœud (*12*, **8**) et considérons tout d'abord le tableau de la Fig. 3B. On balaye le tableau dans le sens des adresses décroissantes à partir de l'adresse **8** . La première donnée rencontrée supérieure à *12* donne l'indice postfixe du père du nœud en question (ici *19*)

Bien entendu, on procéde de manière duale à partir du tableau de la Fig. 3C. Dans ce cas on balaye le tableau dans le sens des adresses croissantes à partir de l'adresse *12* . La première donnée rencontrée inférieure à **8** donne l'indice préfixe du père du nœud en question (ici **7**).

Pour transformer un arbre quelconque en sa représentation informatique, opération appelée codage de l'arbre, on doit tout d'abord procéder à une indexation des nœuds. Pour coder l'arbre sous la forme d'une représentation informatique selon l'invention, on doit indexer les nœuds au moyen des fonctions *Rangprefixe* et *Rangpostfixe* (ou des autres fonctions équivalentes vues plus haut. Sans perte de

généralité, nous limiterons l'exposé de la méthode d'indexation selon l'invention aux deux fonctions précitées.

La méthode d'indexation opère sur une représentation informatique classique de l'arbre au moyen de pointeurs comme, par exemple, celle illustrée en Fig. 2B. Cette représentation informatique classique sera obtenue de manière connue à partir d'un fichier des entrées du dictionnaire. Le chaînage vertical des pointeurs correspond à une relation préfixe des entrées. Le chaînage horizontal des frères d'un même nœud se fait selon l'ordre de primogéniture, tel que celui hérité d'un classement des labels des étiquettes.

On initialise le rang préfixe de la racine à 0 puis on parcourt l'arbre à partir de la racine suivant les pointeurs de fils aîné jusqu'à ce que l'on atteigne une feuille (la plus à gauche selon la convention de représentation choisie ici). Le rang postfixe de cette feuille est initialisé à 0.

Soit maintenant un nœud S de l'arbre ayant pour fils s_0, s_1,..,s_p rangés selon l'ordre croissant de primogéniture (c'est-à-dire ordonnés selon le chaînage horizontal), comme illustré en Fig. 4. On a les relations suivantes :

$$RangPrefixe(s_0) = RangPrefixe(S) + 1$$
$$RangPrefixe(s_{i+1}) = RangPrefixe(s_i) + \Gamma(s_i)$$

$$RangPostfixe(s_{i+1}) = RangPostfixe(s_i) + \Gamma(s_{i+1})$$
$$RangPostfixe(S) = RangPostfixe(s_p) + 1$$
$$\Gamma(S) = \sum_{i=0}^{p} \Gamma(s_i) + 1$$

où $\Gamma(s)$ est la taille du sous-arbre issu de s.

L'indexation par rang préfixe et rang postfixe peut être effectuée en une seule passe, à partir de la racine de l'arbre, en utilisant l'appel récursif d'une fonction qui retourne comme valeur, pour un sommet s donné, la taille $\Gamma(s)$ du sous-arbre qui en est issu. Cette fonction est donnée ci-après en pseudo-code C :

Fonction taille *CodageArbre* (sommet *S*, rang *Prefixe*, rang *Postfixe*, tableau *Bijection*)
Debut

```
TailleSousArbresDesFils=0 ;
Si S est une feuille Alors {
        Bijection[Prefixe]= Postfixe ;
        Retourner 1 ;
}
Sinon pour tous les descendants s de S Faire {
        TailleSousArbre=CodageArbre (s,
                                     Prefixe+1,
                                     Postfixe+TailleSousArbresDesFils,
                                     Bijection) ;
        Prefixe+=TailleSousArbre ;
        TailleSousArbresDesFils+=TailleSousArbre ;
        Bijection[Prefixe]=Postfixe+TailleSousArbresDesFils ;
Retourner TailleSousArbresDesFils+1;
Fin
```

On notera dans le programme ci-dessus que la variable *TailleSousArbre* est la taille du sous-arbre issu du nœud fils (*s*) courant et que la variable *TailleSousArbresDesFils* est la valeur cumulée des tailles des sous-arbres issus des nœuds fils déjà explorés.

La fonction *CodageArbre* crée directement une représentation informatique sous la forme d'un tableau d'index du type de la Fig. 3B, stocké en mémoire. Une fois cette représentation infromatique créée, la représentation initiale par pointeurs, devenue inutile, est supprimée de la mémoire.

Dans la suite, nous nous placerons, par souci de simplification, dans le contexte d'un dictionnaire organisé selon un arbre lexical, chaque entrée du dictionnaire correspondant à une feuille de l'arbre. Comme on vient de le voir, une représentation informatique sous forme de tableau de même taille que celle de l'arbre peut être obtenue par au moyen de la méthode de codage selon l'invention.

Cette représentation informatique est avantageusement utilisée pour rechercher, à partir d'une chaîne de caractères donnée, l'index de l'entrée du dictionnaire correspondante. On prendra conventionnellement comme index le rang préfixe de la

feuille (alternativement on pourrait choisir son rang prefixe inverse). La recherche de l'index fait appel à une première méthode de parcours d'arbre de la racine vers les feuilles selon l'invention.

Réciproquement, cette représentation informatique est avantageusement utilisée pour générer, à partir de l'index d'une entrée du dictionnaire, la chaîne de caractères correspondante. La génération de la chaîne de caractères fait appel à une seconde méthode de parcours d'arbre de la racine vers les feuilles selon l'invention.

On considère d'abord le cas d'une recherche de l'index d'une chaîne de caractères *C*. L'arbre est parcouru à partir de la racine en suivant les arcs dont les étiquettes portent les caractères successifs de *C*.

Avantageusement, la première méthode de parcours d'arbre de la racine vers les feuilles selon l'invention opère de la manière suivante. :

Lors de l'arrivée sur le premier fils s_0 d'un nœud *S*, on initialise :

$$RangPrefixe(s_0)=RangPrefixe(S)+1$$

et l'on calcule la taille $\Gamma(s_0)$ issu de s_0 à partir de :

$$\Gamma(s_0) = RangPostfixe(s_0) - DernierPostfixe$$

où *DernierPostfixe* est le rang postfixe du dernier nœud dont on a abandonné le. parcours du sous-arbre (en d'autres termes, le rang postfixe de la racine du dernier sous-arbre élagué dans le parcours). A titre d'illustration, si en Fig. 3A le nœud de rang postfixe 5 n'a pas été retenu parce que l'arc reliant la racine et ce nœud ne porte pas le caractère recherché, le sous-arbre issu de ce nœud n'est pas parcouru et l'on a *DernierPostfixe* =5. On continue ensuite le parcours par le nœud de rang postfixe 19 et, en cas de succès, par celui de rang postfixe 12. La taille du sous-arbre issu de ce nœud (premier fils s_0 du nœud *S* de rang postfixe 19) est effectivement 7.

On détermine ensuite les rangs préfixes des fils successifs s_i de *S* au moyen des relations de récurrence suivantes :

$$RangPrefixe(s_{i+1}) = RangPrefixe(s_i) + \Gamma(s_i)$$
$$\Gamma(s_{i+1}) = RangPostfixe(s_{i+1}) - RangPostfixe(s_i)$$

A chaque fils exploré s_i, on teste si le caractère en cours c de C est égal au caractère porté par l'étiquette de l'arc joignant S à s_i. Si ce n'est pas le cas on poursuit l'exploration avec le fils suivant s_{i+1} et ainsi de suite jusqu'à ce que l'on ait trouvé le caractère en cours ou exploré tous les fils de S.

Lorsque l'on arrive sur un nœud S donné, on ne connaît pas *a priori* le nombre de ses fils. Pour ce faire, on mémorise avantageusement lors de l'exploration du nœud S la taille $\Gamma(S)$ du sous-arbre qui en est issu. Ensuite, lors de l'exploration successive des fils s_i, on met a jour la variable *TailleSousArbresDesFils* :

$$TailleSousArbresDesFils = \sum_{j=0}^{i} \Gamma(s_j)$$

Et l'on sait que tous les fils s_i de S auront été explorés lorsque :

$$TailleSousArbresDesFils = \Gamma(S) - 1$$

Si tous les fils ont été explorés sans que le caractère en cours n'ait été trouvé, la chaîne de caractère complète C ne correspond pas à une entrée du dictionnaire (une portion de C peut néanmoins en faire partie). Si le caractère en cours a été trouvé pour l'un des fils s_t alors $S=s_t$ et le cycle de recherche reprend avec le caractère suivant. Le processus est itéré jusqu'à ce que l'on atteigne une feuille de l'arbre. L'index recherché est le rang préfixe de cette feuille. Avantageusement, afin de pouvoir rechercher des mots qui sont préfixes l'un de l'autre, comme par exemple « bar » et « barrister » en Fig. 1, on peut ajouter, à la fin de chaque mot un marqueur de fin de mot, par exemple un caractère espace. Dans ce cas, toutes les feuilles de l'arbre portent des marqueurs de fin de mot.

On peut ainsi définir une fonction *ParcoursArbre* qui retourne à partir d'une chaîne de caractères *MotAnalyse*, le rang préfixe de la feuille atteinte au terme du parcours. Cette fonction utilise la représentation informatique de l'arbre selon l'invention. Son pseudo-code C est donné ci-après :

```
Fonction IndexPrefixe ParcoursArbre (chaine MotAnalyse, tableau Bijection)
Debut
    S = Racine;
    Prefixe(S) = 0; // index préfixe de la racine
    TailleSousArbre(S)= Bijection [Prefixe(S)] + 1;
    DernierPostfixe = -1;

    Tant que TailleSousArbre (S) différent de 1 {
        s = fils aîné de S;
        Prefixe(s) = Prefixe(S) + 1;
        TailleSousArbresDesFils = 0;

        Tant que TailleSousArbresDesFils < TailleSousArbre(S) -1;
            Si CaractèreEnCours correspond étiquette de S vers s
            Alors diriger le parcours vers le fils s en faisant
                S=s;
            Sinon { s2 =fils suivant de S ;
                Postfixe(s) = Bijection [Prefixe(s)] ;
                TailleSousArbre(s) = Postfixe(s) - DernierPostfixe;
                Prefixe(s2) = Prefixe(s) + TailleSousArbre(s);
                DernierPostfixe = Postfixe(s);
                TailleSousArbresDesFils += TailleSousArbre(s);
                explorer le fils suivant en faisant
                s=s2
                }
            Retourner CodeParcoursIncomplet
        }
    Retourner Prefixe (S);
```

Réciproquement, si l'on souhaite générer à partir d'un index *I* (supposé égal au rang prefixe d'une feuille de l'arbre) la chaîne de caractères de l'entrée correspondante du dictionnaire, on utilise un seconde méthode de parcours selon l'invention. Cette seconde méthode diffère de la première en ce que la sélection du nœud fils est désormais guidée par la comparaison du rang préfixe de ce nœud avec

l'index I recherché. Plus précisément, le fils s_t est sélectionné dès que la relation suivante est vérifiée :

$$RangPrefixe(s_{t+1})>I$$

L'index I est ainsi approché par valeurs croissantes de rang prefixe des nœuds explorés.

La seconde méthode fait appel au même calcul itératif des rangs prefixe des fils d'un nœud donné S à partir des tailles respectives des sous-arbres $\Gamma(s_i)$. Le critère d'arrêt de l'exploration des fils s_i d'un nœud S donné est également basé sur une comparaison de $\Gamma(S)$ et de la somme des $\Gamma(s_i)$ pour les fils déjà explorés.

On peut définir une fonction *GenerationMot* qui retourne à partir d'un index *GuidePrefixe* la chaîne de caractères correspondante. Cette fonction utilise également la représentation informatique de l'arbre selon l'invention. Son pseudo-code C est donné ci-après :

```
Fonction chaîne GenerationMot (index GuidePrefixe, tableau Bijection)
Debut
    S = racine;
    Prefixe (S) = 0; // index préfixe de la racine
    TailleSousArbre(S)= Bijection [Prefixe(S)] + 1;
    DernierPostfixe = -1;

    Tant que TailleSousArbre(S) différent de 1 {
        s = fils aîné de S;
        Prefixe(s) = Prefixe(S) + 1;
        TailleSousArbresDesFils = 0;

        Tant que TailleSousArbresDesFils < TailleSousArbre(S) - 1 {
            s2 =fils suivant S;
            Postfixe(s) = Bijection [Prefixe(s)];
            TailleSousArbre(s) = Postfixe(s) – DernierPostfixe ;
            Prefixe(s2) = Prefixe(s) + TailleSousArbre(s);
```

```
                Si GuidePrefixe < Prefixe(s2) ;
                Alors {
                        Choisir le parcours vers le fils précédent s en faisant
                        S=s;
                        mettre à jour ChaîneGénérée
                }
                Sinon {
                        DernierPostfixe = Postfixe(s);
                        TailleSousArbresDesFils += TailleSousArbre(s);
                        explorer le descendant suivant en mettant
                        s=s2
                }
        }
        Retourner CodeParcoursIncomplet
}
Retourner ChaîneGénérée;
```

REVENDICATIONS

1) Représentation informatique d'un arbre orienté représentatif de l'organisation d'un ensemble de données, en particulier d'un dictionnaire de données, chaque donnée étant associée à un nœud particulier dudit arbre, caractérisée en ce qu'elle comprend une table de valeurs stockées dans une mémoire, lesdites valeurs étant représentatives des rangs des nœuds dudit arbre ordonnés selon une première relation d'ordre total, les adresses auxqelles lesdites valeurs étant stockées étant représentatives des rangs des nœuds dudit arbre ordonnés selon une seconde relation d'ordre total.

2) Représentation informatique selon la revendication 1, caractérisée en ce que la première relation d'ordre total est une combinaison d'une rel

ation d'ordre de descendance ordonnant un nœud par rapport à ses descendants et d'une relation d'ordre de primogéniture ordonnant les nœuds fils d'un même nœud.

3) Représentation informatique selon la revendication 2, caractérisée en ce qu'un premier nœud de l'arbre est inférieur à un second nœud de l'arbre selon ladite première relation d'ordre total si le second nœud est un descendant du premier nœud ou si, l'ancêtre commun au premier et au second nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon la relation d'ordre de primogéniture.

4) Représentation informatique selon la revendication 2, caractérisée en ce qu'un premier nœud de l'arbre est supérieur à un second nœud de l'arbre selon ladite première relation d'ordre total si le second nœud est un descendant du premier nœud ou si, l'ancêtre commun au premier et au second nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon la relation d'ordre de primogéniture.

5) Représentation informatique selon l'une des revendications 2 à 4, caractérisée en ce que la seconde relation d'ordre total est une combinaison de la relation d'ordre inverse de ladite relation d'ordre de descendance et de ladite relation d'ordre de primogéniture.

6) Représentation informatique selon la revendication 5, caractérisée en ce qu'un premier nœud de l'arbre est inférieur à un second nœud de l'arbre selon ladite seconde relation d'ordre total si le premier nœud est un descendant du second nœud ou si, l'ancêtre commun au premier et au second nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon ladite relation d'ordre de primogéniture.

7) Représentation informatique selon la revendication 5, caractérisée en ce qu'un premier nœud de l'arbre est supérieur à un second nœuds de l'arbre selon ladite seconde relation d'ordre total si le premier nœud est un descendant du second nœud ou si, l'ancêtre commun au premier et au second nœuds ayant un premier fils dont descend le premier nœud ou confondu avec ce dernier et un second fils dont descend le second nœud ou confondu avec ce dernier, ledit premier fils est inférieur audit second fils selon ladite relation d'ordre de primogéniture .

8) Représentation informatique selon l'une des revendications 2 à 7, caractérisée en ce que, les données étant des suites de caractères d'un alphabet muni d'un ordre alphabétique, chaque arc dudit arbre étant associé à un caractère d'au moins une donnée, la relation d'ordre de primogéniture entre deux fils d'un même nœud est donnée par la relation d'ordre alphabétique entre les caractères associés aux arcs respectifs entre ledit nœud et ses deux fils.

9) Méthode de codage d'un arbre orienté représentatif de l'organisation d'un ensemble de données, notamment d'un dictionnaire, chaque donnée dudit ensemble étant associée à un nœud particulier dudit arbre, caractérisée en ce que l'on attribue à chaque nœud dudit arbre un premier et un second index, le premier index étant représentatif du rang du nœud selon une première relation d'ordre total ordonnant les

nœuds dudit arbre, le second index étant représentatif du rang du nœud selon une seconde relation d'ordre total, la première relation d'ordre total étant une combinaison d'une relation d'un ordre de descendance ordonnant un nœud par rapport à ses descendants et d'une relation d'ordre de primogéniture ordonnant les nœuds fils d'un même nœud, la seconde relation d'ordre total étant une combinaison de la relation d'ordre inverse de ladite relation d'ordre de descendance et de ladite relation d'ordre de primogéniture.

10) Méthode de codage selon la revendication 9, caractérisée en ce qu'elle comprend un appel récursif d'une étape de calcul fournissant pour un nœud quelconque de l'arbre, la taille du sous-arbre issu dudit nœud.

11) Méthode de codage selon la revendication 10, caractérisée en ce que, pour un premier fils et un second fils d'un même nœud, dit nœud père, adjacents dans une liste des fils ordonnée selon ladite relation l'ordre de primogéniture, l'étape de calcul détermine le premier index du second fils à partir du premier index du premier fils et de la taille du sous-arbre issu du premier fils, et le second index du second fils à partir du second index du premier fils et de la taille du sous-arbre issu du second fils.

12) Méthode de codage selon la revendication 11, caractérisée en ce que ladite étape de calcul détermine le premier index du fils classé premier dans ladite liste à partir du premier index dudit nœud père et le second index dudit nœud père à partir du second index du fils classé dernier dans ladite liste.

13) Méthode de codage selon l'une des revendications 10 à 12, caractérisée en ce que ladite étape de calcul détermine la taille du sous-arbre issu dudit nœud père à partir de la somme des tailles des sous-arbres issus de ses fils.

14) Méthode de codage selon l'une des revendications 9 à 13, caractérisée en ce qu'elle opère sur une première représentation dudit arbre au moyen de pointeurs dans laquelle, pour un nœud donné, un premier type de pointeur fournit un nœud fils selon la relation d'ordre de descendance et un second type de pointeur fournit la liste de ses autres fils.

15) Méthode de codage selon l'une des revendications 9 à 13, caractérisée en ce qu'elle fournit une table dans laquelle sont rangées des valeurs représentatives desdits premiers index des nœuds dudit arbre en des adresses représentatives desdits seconds index des nœuds dudit arbre.

16) Méthode de codage selon l'une des revendications 9 à 13, caractérisée en ce qu'elle fournit une table dans laquelle sont rangées des valeurs représentatives desdits seconds index des nœuds dudit arbre en des adresses représentatives desdits premiers index des nœuds dudit arbre.

17) Méthode de codage d'une donnée d'entrée appartenant à un ensemble de données organisées selon une structure d'arbre orienté, notamment à un dictionnaire de données, les données étant formées de suites de caractères d'un alphabet muni d'un ordre alphabétique, chaque donnée étant associée à un nœud particulier dudit arbre et à chaque arc dudit arbre étant associé un caractère, caractérisée en ce que, ledit arbre étant représenté au moyen de la représentation informatique selon l'une des revendications 1 à 8, on parcourt l'arbre de nœud en nœud selon un chemin partant de la racine et l'on analyse ladite donnée d'entrée caractère par caractère, le nœud suivant d'un nœud courant dudit chemin étant choisi parmi les fils de ce dernier, le choix étant effectué au moyen d'une succession d'étapes de comparaison, chaque étape de comparaison comparant le caractère en cours de ladite donnée d'entrée et le caractère associé à l'arc reliant le nœud courant à l'un de ses fils, le parcours n'étant interrompu que lorsque la ladite donnée d'entrée est entièrement analysée, la méthode fournissant comme valeur codée de ladite donnée d'entrée un index fonction de l'adresse de la table de ladite représentation informatique représentative du dernier nœud dudit chemin .

18) Méthode de codage selon la revendication 17, caractérisée en ce que ledit index est égal à l'adresse représentative du dernier nœud dudit chemin .

19) Méthode de codage selon la revendication 17, caractérisée en ce que ledit index est égal à la valeur stockée dans ladite table à l'adresse représentative du dernier nœud dudit chemin

20) Méthode de codage selon l'une des revendications 17 à 19, caractérisée en ce que les fils successifs du nœud courant sont déterminés à partir de leurs adresses représentatives respectives dans la table de ladite représentation informatique, l'adresse représentative du fils suivant un fils courant étant obtenue à partir de l'adresse représentative du fils courant et de la taille du sous-arbre issu du fils courant.

21) Méthode de codage selon la revendication 20, caractérisée en ce que la taille du sous-arbre issu du fils courant est obtenue à partir de la valeur stockée dans ladite table à l'adresse représentative du fils courant et de la valeur stockée dans ladite table à l'adresse représentative du fils précédent.

22) Méthode de décodage d'un index représentatif d'une donnée appartenant à un ensemble de données organisées selon une structure d'arbre orienté, en particulier d'un dictionnaire de données, les données étant formées de suites de caractères d'un alphabet muni d'un ordre alphabétique, chaque donnée étant associée à un nœud particulier dudit arbre et à chaque arc dudit arbre étant associé un caractère, caractérisée en ce que, ledit arbre étant représenté au moyen de la représentation informatique selon l'une des revendications 1 à 8, on parcourt l'arbre selon un chemin partant de la racine, le nœud suivant un nœud courant dudit chemin étant choisi parmi les fils de ce dernier, le choix étant effectué au moyen d'une succession d'étapes de comparaison, chaque étape de comparaison comparant ledit index à un index représentatif de l'un desdits fils dans ladite représentation informatique, la méthode fournissant comme donnée décodée la chaîne des caractères associés aux arcs formant ledit chemin.

23) Méthode de décodage selon la revendication 22, caractérisée en ce que ledit index représentatif est une adresse dans ladite table de la représentation informatique.

24) Méthode de décodage selon la revendication 22, caractérisée en ce que ledit index représentatif est une valeur stockée dans ladite table de la représentation informatique.

25) Méthode de décodage selon l'une des revendications 22 à 24, caractérisée en ce que les fils successifs du nœud courant sont déterminés à partir de leurs adresses représentatives respectives dans la table de ladite représentation informatique, l'adresse représentative du fils suivant un fils courant étant obtenue à partir de l'adresse représentative du fils courant et de la taille du sous-arbre issu du fils courant.

26) Méthode de codage selon la revendication 25, caractérisée en ce que la taille du sous-arbre issu du fils courant est obtenue à partir de la valeur stockée dans ladite table à l'adresse représentative du fils courant et de la valeur stockée dans ladite table à l'adresse représentative du fils précédent.

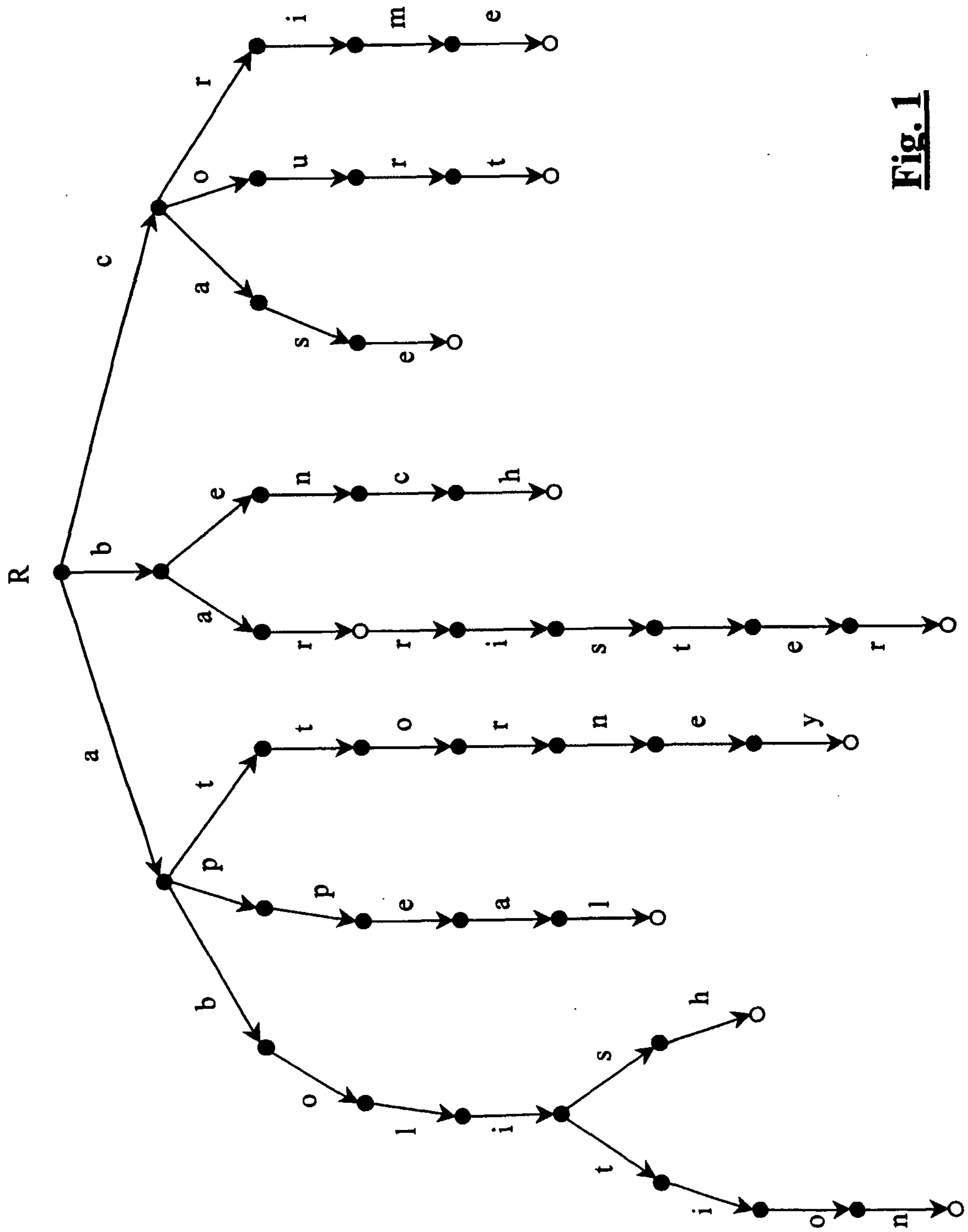
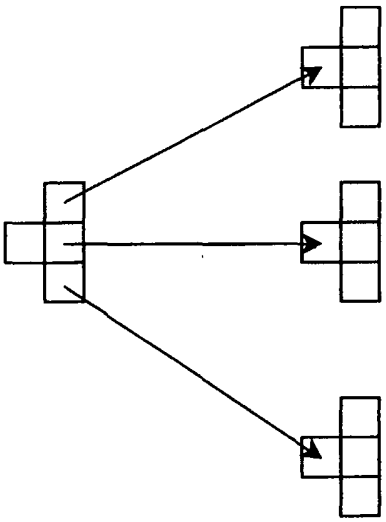


Fig. 1

Fig. 2A



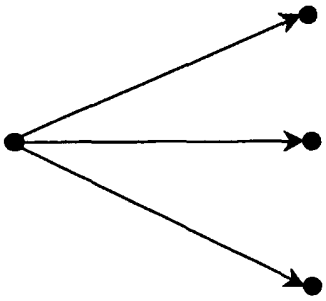
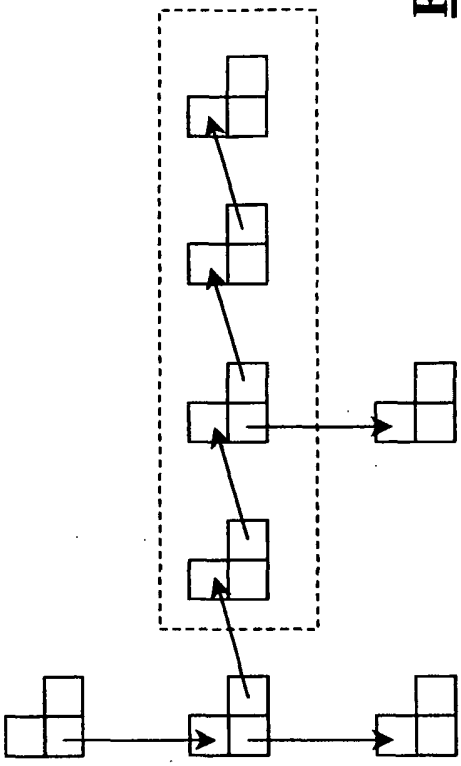


Fig. 2B



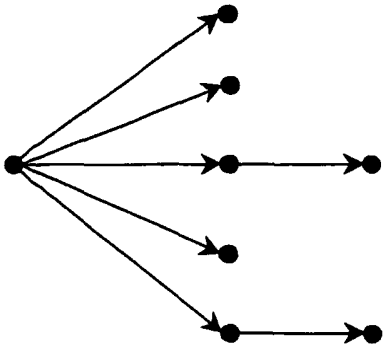


Fig. 3A

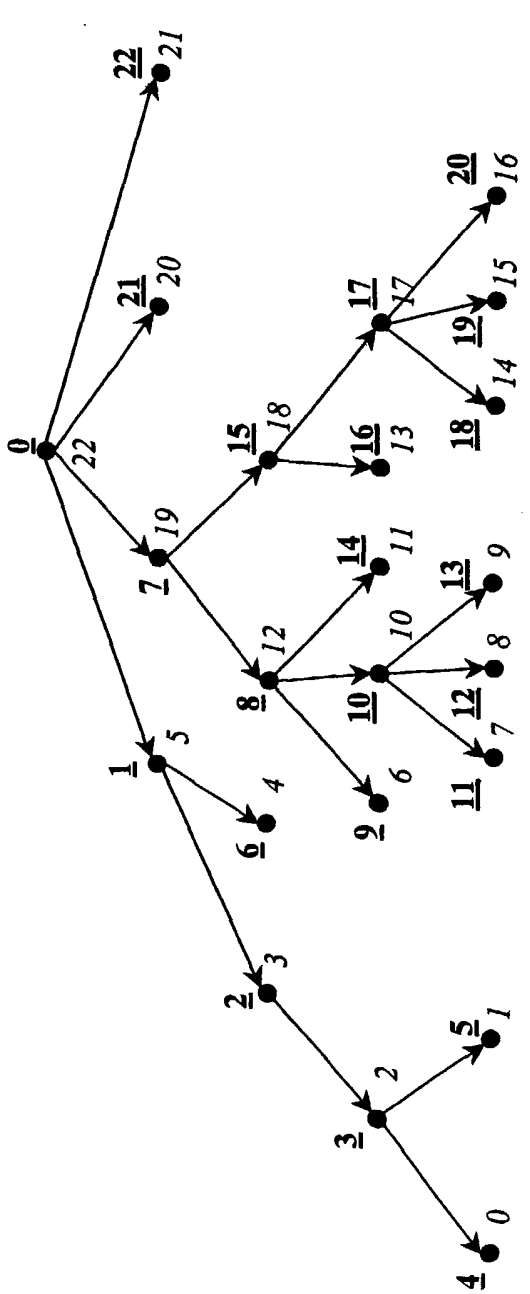


Fig. 3B

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	22	5	3	2	0	1	4	19	12	6	10	7	8	9	11	18	13	17	14	15	16	20	21

Fig. 3C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T^{-1}	4	5	3	2	6	1	9	11	12	13	10	14	8	16	18	18	20	17	15	7	21	22	0

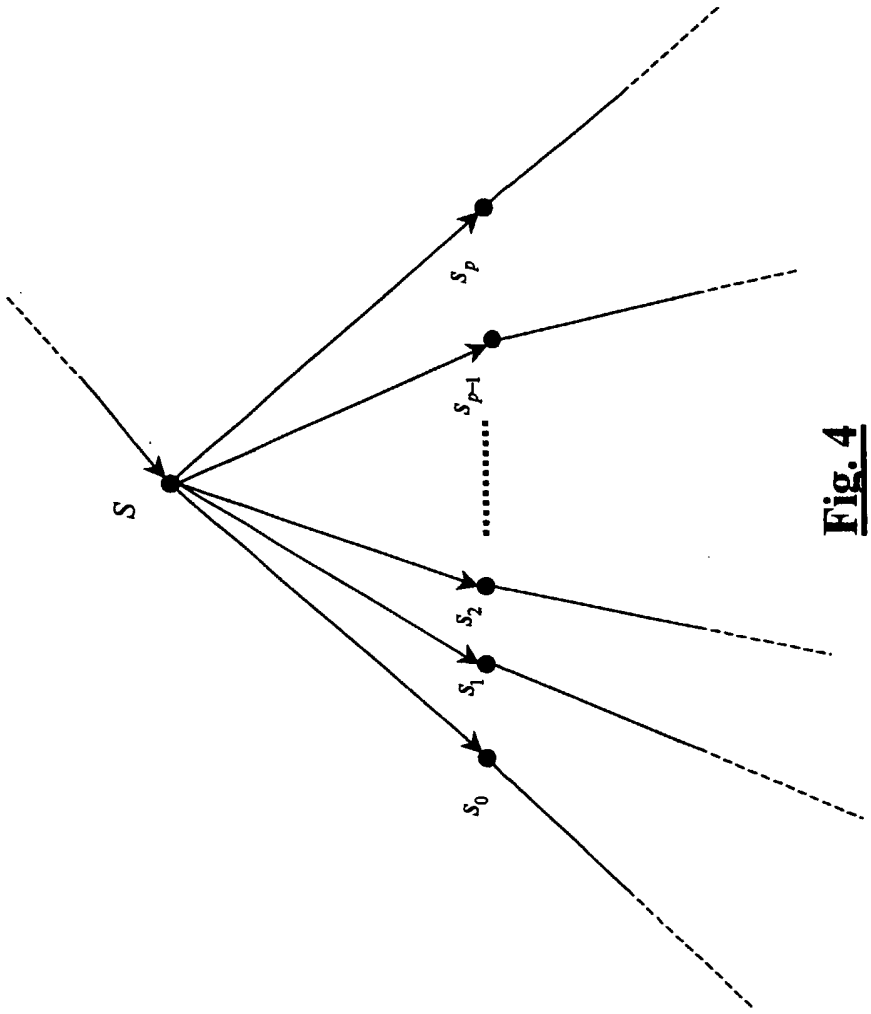


Fig. 4