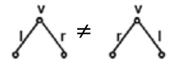
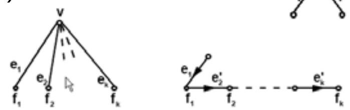
Arbres, répertoires arborescents

(KNUTH 97 Vol 1 § 2.3)

Arbres

- avec racine et dirigé
- arcs (branches) ordonnés





- arbre multi-branches → arbre binaire.
  a) arbre nul
  b) Racine
  c) racine avec 1 arbre binaire de gauche et 1 arbre binaire de droite.
- définition récursive ⇒ algorithme récursifs pour le traitement des arbres (algorithmes itératifs sont moins coûteux)

Arbres

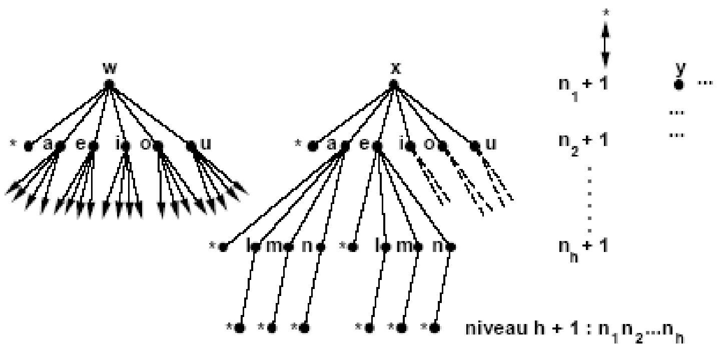
- Arbre logique $\triangleq$ description de l'arbre, sans aucune information sur sa représentation en mémoire
- $\neq$ arbre physique où la structure en mémoire est donnée
- Les clés possibles sont générées par une grammaire G_K, dont un ss-ensemble K de clés sont présentes
- p. 187 et 188 où * < b < a, ... Séquence de permutation π_K pour K

1. Variétés d'arbres

- noeud de caractère : donnée associée = 1 caractère
- noeud de clé : donnée associée = 1 clé.
- arbre positionnel ou digital : tous les noeuds = nœuds de caractères
- index multidimensionnel : chaque nœud a un arc sortant pour chaque caractère possible $\in$ alphabet

1. Variétés d'arbres

- Exemple



1. Variétés d'arbres

- Nombre de feuilles = $1+\sum_{j=1}^{h}\prod_{i=1}^{j} n_i$
- Nombre de noeuds = $1+2\sum_{j=1}^{h}\prod_{i=1}^{j} n_i$
- Autrement : nombre de noeuds =

$$n_1+1+n_1(n_2+1)+...+n_1n_2..n_{h-1}(n_h+1)$$

$$+n_1n_2...n_h = 2\sum_{j=1}^{h}\prod_{i=1}^{j} n_i + 1 \simeq 2*\text{nb de feuilles}$$

- Dans un index multidimensionnel, la chaîne de caractères associée à une feuille = 1 clé possible
- Dans un noeud terminal (feuille) associé à une clé présente : pointeur vers une zone attributs ≠ pointeur nul pour une clé possible mais absente

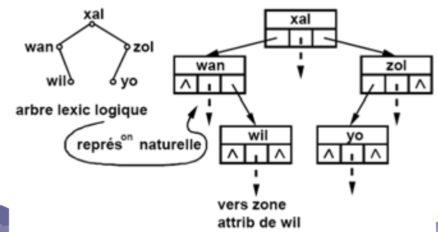
1. Variétés d'arbres

- Recherche d'une clé yon : dans nœud associé à 0, doit-on choisir un pointeur vers une zone attributs de yo ou un pointeur pour continuer la recherche ?
- 2 solutions :
  - 2 pointeurs distincts dans chaque noeud : 1 pour la zone attributs et 1 pour la recherche à continuer
    - 2 N pointeurs + N caractères
    - N nœuds
  - 1 caractère terminaison * : pointeur vers la zone attributs s'il est associé à *, pour continuer la recherche sinon.
    - 2 N (pointeur + caractère)
    - 2 N noeuds

1. Variétés d'arbres

- Considérons l'arbre avec des noeuds clé :
  - arbre binaire lexicographique ≜ arbre construit en prenant la première clé comme racine puis les clés successives attachées au sous arbre de gauche ou de droite selon qu'elle est inférieure ou supérieure à racine

Ex <xal, wan, wil, zol, yo>



1. Variétés d'arbres

- ∃ une transformation simple pour convertir tout arbre multi-voie avec des arcs ordonnés en un arbre binaire à racine
- Exemple : Figures 6.3 et 6.6

2. Opérations sur les arbres

- recherche, insertion, suppression, traversée/énumération (par exemple : traiter chaque noeud)

a) recherche et traversée dans arbre binaire lexicographique

- recherche d'une entrée : l'algorithme découle de la définition récursive :
  - pas d'arbre, entrée absente
  - comparer la clé avec la racine
    - = : localisée
    - < : sous-arbre de gauche
    - > : sous-arbre de droite

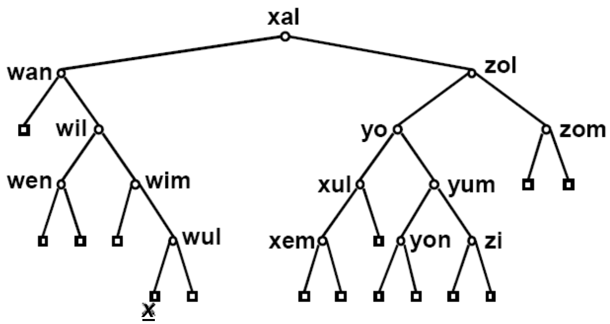
2. Opérations sur les arbres

- arbre à N nœuds ; $l_i \triangleq$ niveau de l'entrée i
- Pour clé présente, si les N entrées sont également probables : $E_p(N) = \frac{1}{N}\sum_{i=1}^{N} l_i$
- Si la distribution de $\mathbb{P}$ est non uniforme et $p_i \triangleq$ fréquence observée pour clé k_i : $E_p(N) = \frac{1}{W}\sum_{i=1}^{N} l_i p_i$
- Où
  - W $\triangleq \sum_{i=1}^{N} p_i$ = constante de normalisation

2. Opérations sur les arbres

- Cas de recherches infructueuses.
  - Recherche infructueuse $\rightarrow$ insertion d'une nouvelle entrée, c'est-à-dire une nouvelle feuille
  - Pour représenter toutes les recherches infructueuses, "extension" de l'arbre en attachant une paire de feuilles "vacantes" à toutes les feuilles initiales et une feuille vacante à tous les noeuds internes de degré 1 $\rightarrow$ arbre étendu a 2N arcs et 2N+1 noeuds (c'est-à-dire adjonction de N+1 noeuds vacants)
  - Chaque noeud vacant représente l'ensemble de toutes les clés comprises entre 2 clés présentes dans l'arbre initial

2. Opérations sur les arbres



- Si x est une clé possible dans noeud ***X*** :
  - wim < x < wul

2. Opérations sur les arbres

- Les clés sont classées lexicalement k_1 k_2 ... K_N. Soient
  - $q_0 \triangleq$ Pr{la clé recherchée précède k_1}. Constante (W)
  - $q_i \triangleq$ Pr{la clé recherchée est comprise entre k_i et k_{i+1}}. Constante (W)
  - $q_N \triangleq$ Pr{la clé recherchée > k_N}. Constante (W)
  - $l'_j \triangleq$ niveau du noeud vacant avec une fréquence q_j.

$$E(N) = \frac{1}{W}\left[\sum_{i=1}^{N} p_i l_i + \sum_{j=0}^{N} q_j (l'_j - 1)\right]$$ À minimiser

2. Opérations sur les arbres

- Où la constante de normalisation vaut :

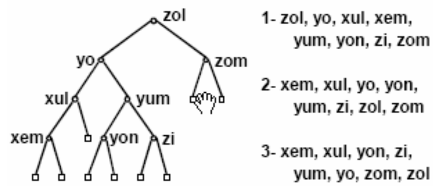
$$W = \sum_{i=1}^{N} p_i + \sum_{j=0}^{N} q_j \qquad \text{(poids de l'arbre)}$$

  – ou minimiser:
    - t_{rech} max
    - t_{insert}
    - t_{suppr}

2. Opérations sur les arbres

- Traversée

1. préordonnée (notation préfixée des expressions algébriques)
   – Visiter la racine
   – Traverser le sous-arbre de gauche en séquence préordonnée
   – Traverser le sous-arbre de droite en séquence préordonnée



2. Opérations sur les arbres

2. symétrique ($\rightarrow$ lexicograhique)
   - traverser le sous-arbre de gauche en séquence symétrique
   - visiter la racine
   - traverser le sous-arbre de droite en séquence symétrique
3. postordonnée (notation postfixée)
   - traverser le sous-arbre de gauche en séquence postordonnée
   - traverser le sous-arbre de droite en séquence postordonnée
   - Visiter la racine

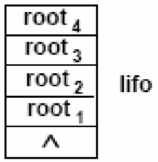
! Un arbre binaire n'est pas défini de façon non ambiguë par l'une de ces 3 listes (condition pour la définition dans KNUTH)

2. Opérations sur les arbres

- Algorithme pour énumération symétrique
- lifo $\triangleq$ pile de pointeurs
- root $\triangleq$ variable pointeur $\rightarrow$ racine de l'arbre dont les noeuds ont la forme :
  - comp node =
    - <node-ptr llink,string key,node-ptr rlink>

2. Opérations sur les arbres

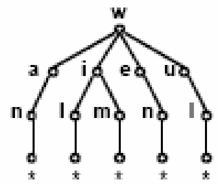
```
procedure symm_list(root);
   node-ptr root, ptr-stack lifo;
      ~~Lifo ⇐ ∧~~ ;    Push(LIFO, ∧)
   do while root ≠ ∧  ∨  TOP(lifo) ≠ ∧;
      do while root ≠ ∧;
         PUSH(lifo,root);
         root ⇐ llink(@root);
      end
      root ⇐ TOP(lifo);POP(lifo);
      if root ≠ ∧ then
      do
         PRINT key(@root);
         root ⇐ rlink(@root);
      end
   end
end symm_list
```



2. Opérations sur les arbres

b) Traversée et recherche dans des arbres multivoies

- depth first
  - w,a,n,*,i,l,*,m,*,e,n,*,u,l,*
- breadth first
  - w,a,i,e,u,n,l,m,n,l,*,*,*,*,*



2. Opérations sur les arbres

- $\overline{t}_{rech}$ dans un arbre multi-voie avec N noeuds

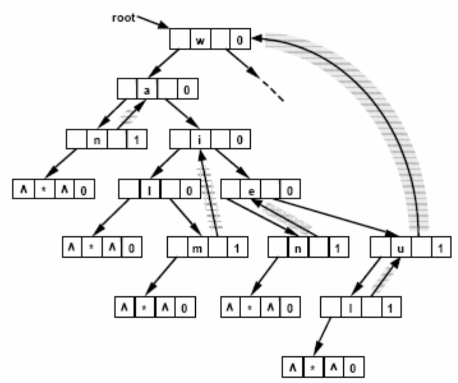
$$T(N) = a.h + b \sum_{j=1}^{h} f(n_j)$$

- où :
  - h = nombre de niveaux
  - n_j = nombre de noeuds au niv j
  - a = temps pour aller d'un niveau au suivant
  - $f(n_j)$ = fonction dépendant de la méthode de recherche dans un même niveau
  - b = facteur caractérisant $t_{accès}$ à 1 noeud dans un niveau

2. Opérations sur les arbres

- Recherches généalogiques et chaînage enfilé
- Chaînage unidirectionnel ⇒ difficile de retrouver le père et les ancêtres (même problème que le prédécesseur dans une liste chaînée simple). Les pointeurs supplémentaires pour établir la relation de filiation constituent un "chaînage enfilé" → arbre enfilé
- Un arbre binaire de N noeuds a de la place pour 2N pointeurs dont (N-1) sont utilisés pour spécifier les arcs. Il reste (N+1) pour nous servir

2. Opérations sur les arbres



2. Opérations sur les arbres

- llink pointe vers le descendant
- rlink pointe vers
  - l'aîné des petits frères si flag = 0
  - le père si flag = 1 (pour cadet d'une génération)

- On peut ainsi remonter vers n'importe quel ancêtre et n'importe quel frère aîné

3. Arbres positionnels

- Dans un index multidimensionnel, (le nombre de comparaisons) le temps pour trouver une clé E(N) = a.h où
  - a = une constante
  - h = le nombre de caractères dans la clé
- a $\leftrightarrow$ opération d'indexation est peu élevé
- Si n_i caractères sont possibles à la i[e] position de la clé et la clé est de longueur $\leq$ h, le nombre de clés possibles est

$$N = \sum_{j=1}^{h} \prod_{i=1}^{j} n_i$$

3. Arbres positionnels

- Si $\forall i \forall j \quad n_i \simeq n_j \simeq n, N \simeq \prod_{i=1}^{h} n_i$

$$(N \simeq \sum_{m=1}^{h} n^m = \frac{n^{h+1} - 1}{n - 1} - 1 \simeq n^h) \qquad si \quad n \gg 1$$

- Notant par γ la moyenne géométrique $(\gamma^h = \prod_{i=1}^{h} n_i)$

$$N \simeq \gamma^h \quad ou \quad h \simeq \log_\gamma N$$

- c'est-à-dire que dans index multidimensionnel, $t_{accès} \div \log N$

3. Arbres positionnels

- Arbre de la Briandais
  - ≜ arbre positionnel où un chemin racine feuille ↔ une clé c'est à dire le nombre de feuilles = nombre de clés présentes (élagage de l'index multidimensionnel)
- Version binaire plutôt que multi-voie (figure 6.10 p. 201)
- noeud : comp node =

  <node ptr match, str info, node ptr mismatch>

3. Arbres positionnels

- Algorithme de recherche en donnant les pointeurs root et key
-
```
procedure tree_search1(root,key,index,place);
int index;node-ptr root, place;
string key; boolean searching;
   place ⇐ root; index ⇐ 1; searching ⇐ place ≠ ∧;
   do while searching;
      if info(@ place) = key[index] then
      do
        if key[index] ≠ *
           then place ⇐ match(@ place);
           else searching ⇐ false;
        index ⇐ index+1;
      end
      else
      do
        if mismatch(@ place) = ∧
        then searching ⇐ false;
        else place ⇐ mismatch(@ place);
      end
   end
end tree_search1
```

cf fig. 6.10

3. Arbres positionnels

- Le pointeur place pointe vers le noeud où la recherche s'est terminée.

- Index
  - ρ(key)+1 si succès
  - Position du 1er caractère ≠

- Représentation en tableau d'un arbre de la Briandais (table 6.1 p. 203).

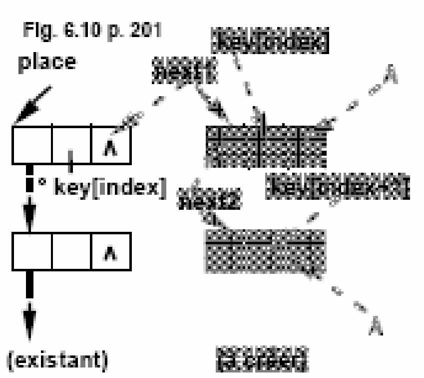
- *Algorithme d'insertion en donnant les pointeurs root et key*

3. Arbres positionnels

- procedure tree_insert(root,key,place);

```
   int index;  node-ptr root, place, next;  boolean adding; string key;
   call tree_search1(root,key,index,place);
   if index = ρ(key)+1 then return;
   CREATE(node,next);  mismatch(@ place) ⇐ next;
   adding ⇐ key[index-1] ≠ *;
   do while adding;
       mismatch(@ next)⇐ ∧;   info(@ next) ⇐ key [index];   place ⇐ next;
       index ⇐ index + 1;
       if key[index-1] ≠ * then
       do
          CREATE(node,next); match(@ place) ⇐ next;
       end
       else adding ⇐ false;
   end
end tree_insert
```

3. Arbres positionnels



3. Arbres positionnels

- à la sortie de tree-insert, place pointe vers noeud ⊃ le pointeur vers la zone attributs de la clé insérée
- La suppression est plus complexe. On peut supprimer un noeud seulement quand son pointeur mismatch = ∧. Comme la suppression commence par le dernier caractère, le chemin doit être mémorisé temporairement ou en chaînage enfilé (⇒ mise à jour des pointeurs pour insertions !)

3. Arbres positionnels

- Soient
  - $n_i \triangleq$ nombre moyen de noeuds au niveau i
  - $\overline{h} \triangleq$ nombre moyen de niveaux
- Distribution uniforme ⇒ nombre moyen de comparaisons au niveau i = $\frac{1}{n_i}\sum_{j=1}^{n_i} j = \frac{n_j+1}{2}$
- Si ∀i ∀j $n_i \simeq n_j$, le nombre moyen de comparaisons

$$E_p(N) \simeq \sum_{i=1}^{\overline{h}} \frac{1+n_i}{2} = \overline{h}\frac{1}{2} + \frac{1}{2}\sum_{i=1}^{\overline{h}} n_i = \overline{h}\frac{1+\overline{n}}{2} \qquad \text{ou} \quad \overline{n} \triangleq \frac{1}{\overline{h}}\sum_{i=1}^{\overline{h}} n_i$$

- d'où $t_{rech} = O(\overline{n}.\overline{h})$

3. Arbres positionnels

- $N \simeq \overline{n}^{\overline{h}} \Rightarrow h \simeq \log_{\overline{n}} N$
- c'est-à-dire $O(\overline{n}\log_{\overline{n}} N) \overset{*}{>} a \log_{\gamma} N$ de l'index multidimensionnel car la recherche est linéaire/indexation.

$*\left\{\right.$
  - n_i peut être très ≠ de n_j
  - Pour certains chemins, le nombre de nœuds peut être très ≠ du nombre moyen de n_i (par exemple : le seul successeur de q est u)

- Pour accélérer :
  - liste filiale ordonnée alphabétiquement ⇒ entrée absente dès que la valeur du noeud est excédée
  - si la distribution est non uniforme, les fréquences sont élevées au début de l'arbre : fig. 6.11 (optimal) comparée à 6.10

3. Arbres positionnels

c) Tries (retrieval)

$\triangleq$ arbre positionnel dans lequel l'ensemble descendants d'un caractère correspond à un vecteur de pointeurs dont chaque composant pointe vers un descendant

Recherche par opération d'indexation $\Rightarrow t_{rech} = O(a.h)$ où a est constante et petite c'est-à-dire $\simeq$ index multidimensionnel

3. Arbres positionnels

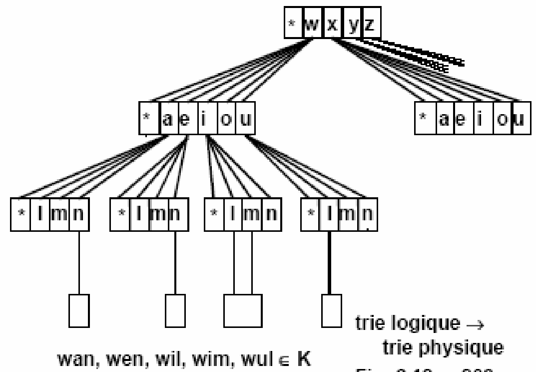


Fig. 6.13 p. 208

3. Arbres positionnels

- indexation $\Rightarrow$ 27 pointeurs à chaque niveau si les clés sont formées par des caractères alphabétiques. Pour les clés générées par G_K, une table simple (table 6.2 p. 209) permet de réduire à 6 pointeurs ($\leftrightarrow$ le plus grand ensemble de descendants)
- Recherche dans un trie
- comp node = <node-ptr array p[1:6]>

3. Arbres positionnels

- procedure trie_search(root,key,place);

```
int i,j; node-ptr root,place;
int-array num[1:27];string key;
i⇐ 1;
j ⇐ num[key[1]];
do while key[i] ≠ * ∧ p[j](@root)≠ ∧
   root ⇐ p[j](@root);
   i ⇐ i+1;
   j ⇐ num[key[i]];
end;
place p[1](@root);
end trie_search
```

donne n° d'ordre +1 dans l'alphabet (table 6.2)

3. Arbres positionnels

- Comme indiqué sur la figure 6.13, place pointe, à la sortie du programme, vers la zone des attributs de la clé si elle est présente, place = $\wedge$ sinon (tous les pointeurs sont initialisés à $\wedge$)

4. Arbres lexicographiques

KNUTH 98 Vol. 3 § 6.2 et 6.3

- Insertions séquentielles des clés selon un aiguillage binaire. Deux observations :
  - construire l'arbre lexicographique de K $\Leftrightarrow$ trier K car une traversée symétrique de l'arbre $\rightarrow$ liste triée lexicographiquement. "treesort"
  - si K est trié au départ et les insertions des membres se font selon l'ordre d'une recherche binaire, l'arbre est complet (c'est-à-dire il y a des feuilles uniquement aux 2 derniers niveaux : $\forall$ noeud, la $\neq^{ce}$ des ordres des sous-arbres de droite et de gauche $\leq$ 1)

4. Arbres lexicographiques

- Exemple : construire l'arbre lexicographique complet de la Figure 6.14

  <wan,wen,wil,wim,wul,xal,xem,xul,yo,yon,yum,zi,zol,zon>

  – $\lfloor(l+f)/2\rfloor$ et $\lceil(l+f)/2\rceil$ $\Rightarrow$ arbre complet unique seulement si complètement rempli à tous les niveaux
    - $\lfloor\ \rfloor = (l+f)/2 = \lceil\ \rceil$

a) Arbre optimal avec clés également pondérées.

  – Si N est donné, l'arbre binaire complet a une hauteur minimale. Si les clés sont également pondérées, l'arbre optimal ($\leftrightarrow t_{rech}$ min) doit être complet. (S'il contient un sous-arbre non complet, ce dernier peut être remplacé par un sous-arbre complet avec t_{rech} moindre). L'arbre optimal ayant tous ses sous-arbres complets est lui-même complet

4. Arbres lexicographiques

- Soit $B_h \triangleq$ arbre binaire complètement rempli de hauteur h

- N = nombre de noeuds = $1+2+2^2+ \ldots + 2^{h-1} = 2^h-1$
  $\Rightarrow h = \log(N+1) \simeq \log N$

- $I(N) \triangleq$ longueur totale des chemins pour les N nœuds

$$I(N) = 1 + 2.2 + 3.2^2 + \ldots + h.2^{h-1}$$

$$\neq^{ce} \quad 2\,I(N) = 1.2 + 2.2^2 + 3.2^3 + \ldots + h.2^h$$

$$I(N) = h.2^h-(2^{h-1}+2^{h-2}+ \ldots +2+1) = 2^h(h-1)+1$$

4. Arbres lexicographiques

$$E_p(N) = \frac{I(N)}{N} = \frac{2^h(h-1)+1}{2^h-1} = \frac{(h-1+2^{-h}(1+2^{-h})}{(1-2^{-h}(1+2^{-h})}$$

$$\simeq h - 1 + \frac{h}{2^h} \simeq \log(N+1) - 1 + \frac{\log N}{N}$$

$$\left(h = 3 \left\{ \begin{array}{c} 2^{-h} = \frac{1}{8} \\ 2^{-2h} = \frac{1}{64} \end{array} \right. \right)$$

- Max = h $\simeq E_p(N) + 1$ (car $\simeq$ moitié des nœuds au niveau h)

4. Arbres lexicographiques

b) Arbre lexicographiques binaires aléatoires

- Si clés sont insérées aléatoirement, on a des branchements plus ou moins longs. Le pire : quand on insère les clés selon l'ordre lexicogr. $\rightarrow$ une chaîne de N entrées et

$$\bar{t}_{rech} = O\left(\frac{N+1}{2}\right)$$

- Un arbre lexicographique $T_j \leftrightarrow$ permutation π_j de {1,2,...,N} d'où N! d'arbres possibles

(pas tous $\neq^{ts}$: 

$\leftrightarrow$ {y,x,z} et {y,z,x})

4. Arbres lexicographiques

- Soit l'arbre binaire étendu $T_j \leftrightarrow \pi_j(N)$
  - $l_j(N) \triangleq \sum$ longueur des chemins sur tous les nœuds internes correspondant
  - $L_j(N) \triangleq \sum$ longueur des chemins sur tous noeuds externes (ou feuilles) correspondant
  - $l_j(N) = \sum^{N}_{i=1} l_i(Tj)$ où $l_i \triangleq$ niveau du nœud interne i
  - $L_j(N) = \sum^{N}_{i=0} [l'_i(T_j) - 1] \triangleq$ niveau du nœud externe i

$$E_p(N) = \frac{l_j(N)}{N} \quad et \quad E_a(N) = \frac{l_j(N)}{N+1}$$

4. Arbres lexicographiques

- Moyenne sur les N! arbres :
  - $E_p(N) = \frac{1}{N!}\sum_{j=1}^{N!}\frac{l_j(N)}{N} = \frac{l(N)}{N.N!} \quad avec \quad l(N) \triangleq \sum_{j=1}^{N!} l_j(N)$
  - $E_a(N) = \frac{1}{N!}\sum_{j=1}^{N!}\frac{L_j(N)}{N+1} = \frac{L(N)}{(N+1)!} \quad avec \quad L(N) \triangleq \sum_{j=1}^{N!} L_j(N)$
- Dans un arbre étendu, tout sous arbre est étendu c'est-à dire si un sous arbre a n noeuds internes, il a (n+1) feuilles $\Rightarrow$ si la racine de ce sous arbre (un noeud interne de l'arbre initial) contribue pour m dans calcul de $l_j(N)$, elle contribue pour (m+1) dans calcul de $L_j(N) \Rightarrow L_j(N)$ et $l_j(N)$ diffèrent de 1 par noeud interne

4. Arbres lexicographiques

- ⇒ $L_j(N)$ et $l_j(N)$ diffèrent de 1 par noeud interne

$$\Rightarrow l_j(N) = L_j(N) - N$$

$$\Rightarrow \sum_{j=1}^{N!} l_j(N) = \sum_{j=1}^{N!} L - j(N) - N.N!$$

- Par substitution : $e_p(N) = \frac{N+1}{N} E_a(N) - 1$
- Exemple : Figure 6.7
  - $l_K(N)$ = 1+2.2+3.3+4.4+4.5 = 50
  - $L_K(N)$ = 1.2+2.3+4.4+8.5 = 64
  - et $L_K(N) - l_K(N)$ = 14 = N

4. Arbres lexicographiques

- Nombre de comparaisons pour trouver une clé = 1 + nombre de comparaisons pour l'insérer

$$\Rightarrow E_p(N) = 1 + \frac{E_a(0) + E_a(1) + ... + E_a(N-1)}{N}$$

$$\Rightarrow (n+1)E_a(N) = 2N + E_a(0) + ... + E_a(N-1)$$

$$\Rightarrow NE_a(N-1) = 2(N-1) + E_A(0) + ... + E_a(N-2)$$

$$\Rightarrow (N+1)E_a(N) - NE_a(N-1) = 2 + E_a(N-1)$$

C'est-à-dire $E_a(N) = E_a(N-1) + \frac{2}{N+1}$

$$E_a(0) = 0 \Rightarrow E_a(N) = 2\left(\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N-1}\right) = 2H_{N+1} - 2$$

où $H_N \triangleq$ série harmonique = $\sum_{i=1}^{N} \frac{1}{i}$

4. Arbres lexicographiques

- Or $\lim_{N\to\infty} H_N = \ln N + \gamma$ où γ = 0.57722 (c^{te} d'Euler)
- $E_a(N) \simeq 2\ln(N{+}1){+}2\gamma{-}2 \simeq 1.386\log N{-}0.85$
- $E_p(N) = \frac{N+1}{N}E_a(N){-}1 \simeq 2\ln(N{+}1){+}2\gamma{-}3 \simeq 1.386\log N{-}1.85$
- $E_p(N) \simeq \log N$ pour un arbre complet $\Rightarrow \bar{t}_{rech}$ pour un arbre construit aléatoirement est seulement $\simeq$ 40% plus élevé

4. Arbres lexicographiques

- Le pire : $E_p(N) = \frac{(N+1)}{2}$ mais en moyenne, le comportement n'est pas trop ≠ d'un arbre complet !
- Algorithme 6.5 : recherche dans un arbre lexicographique $\rightarrow$ insertion

4. Arbres lexicographiques

- La suppression est plus compliquée :
  - supprimer une feuille : mettre 1 pointeur à $\wedge$.
  - supprimer un noeud qui n'a qu'un seul sous-arbre, soit de droite, soit de gauche (Fig. 6.7) :
    - wan supprimé en mettant val de son rlink dans llink de xal
    - xul supprimé en mettant val de son llink ds llink de yo
  - Supprimer un noeud qui a les 2 sous-arbres : le remplacer par son successeur lexical (c'est-à-dire le premier noeud rencontré dans une traversée symétrique du sous-arbre de droite). Ce successeur a toujours son llink = $\wedge$ $\Rightarrow$ à son déplacement, on doit juste rattacher son sous arbre de droite éventuel à son ascendant

4. Arbres lexicographiques

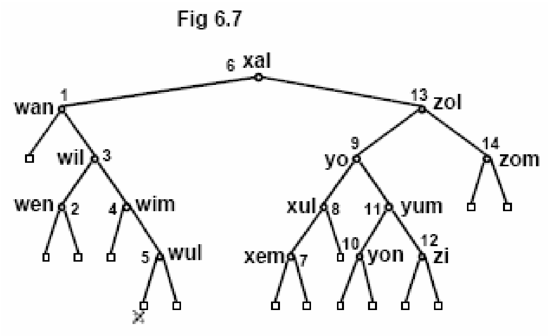
- Exemple :
  - dans la Figure 6.7, on supprime xal en le remplaçant par xem, son successeur. Comme xem est une feuille, il n'y a rien d'autre à faire
  - dans la Figure 6.5, on supprime wil en le remplaçant par son successeur xal qui lui-même sera remplacé par yo, la racine de son sous-arbre de droite
- Nombre de comparaisons pour construire l'arbre :

$$E_a(0)+E_A(1)+...+E_a(N-1) \simeq \sum_{j=1}^{N-1} [2\ln(j+1)+2\gamma-2]$$

$$= O(\ln N!) = O(N \log N)$$

$\Rightarrow$ $t_{exéc}$ $\simeq$ pour treesort, quicksort et tri par fusion

4. Arbres lexicographiques



5. Arbres sous contrainte(s)

- Pour un arbre aléatoire, $\overline{t}_{rech}$ = O(1.4 log N) mais t_{rech}max = O(N). Pour prévenir le cas extrême où h est élevée, introduire des contraintes

a) Arbres équilibrés en hauteur

$\triangleq$ arbre dans lesquels, pour tout noeud, la différence de hauteur entre les sous-arbres de gauche et droite ≤ 1

5. Arbres sous contrainte(s)

- Pour un nombre donné de noeuds N, la hauteur minimale est celle de B_h, pour un arbre complètement rempli : $N = 2^h-1$

$\Rightarrow h_{min} = \log(N+1)$. ($B_h$ est équilibré en hauteur)

- h_{max} = ? dans un arbre équilibré en hauteur de N nœuds $\leftrightarrow$ nombre minimum de noeuds dans un arbre équilibré en hauteur de hauteur h ?
  - Soit F_h l'arbre équilibré en hauteur de hauteur h ayant comme minimum noeuds $F_h = < F_l, v, F_r >$.
  - Supposons $F_l = F_{h-1} \Rightarrow F_r = F_{h-2}$ pour minimiser le nombre noeuds. Notant $|F_h| = N$ l'ordre de l'arbre : $|F_h| = 1 + |F_{h-1}| + |F_{h-2}|$

5. Arbres sous contrainte(s)

- Partant de F_1 et F_2 avec 1 et 2 noeuds $\rightarrow$ arbre de Fibonacci
- Suite de Fibonacci $f_{n+2} = f_{n+1} + f_n$; $n \geq 0$; $f_0=0$ et $f_1=1$
  - 0,1,1,2,3,5,8,13,21,...
- $|F_1| = f_3 - 1$ et $|F_h| = f_{h+2} - 1 = N$
- Φ, nombre d'or $\triangleq \frac{1}{2}(1+\sqrt{5}) \simeq 1.618$ $\quad et \quad$ $\overline{\Phi} = 1-\Phi = \frac{1}{2}(1-\sqrt{5})$

  $f_n = \frac{1}{\sqrt{5}}(\Phi^n - \overline{\Phi}^n)$ (par méthode matricielle sur $\begin{bmatrix} f_{n+1} \\ f_n \end{bmatrix}$)

  $\Rightarrow \frac{\Phi^h}{\sqrt{5}} - 1 < f_h < \frac{\Phi^h}{\sqrt{5}} + 1$

5. Arbres sous contrainte(s)

$$\frac{\Phi^{h+2}}{\sqrt{5}} - 1 < f_{h+2} = N + 1$$

$$(h+2)\log\Phi < \log(N+2) + \log\sqrt{5}$$

$$\Rightarrow h_{max} < 1.44\log(N+2) - 0.33$$

Log N < h < 1.44 log(N + 2)

$t_{rech} \simeq logN + c$ avec $c \simeq 0.25$

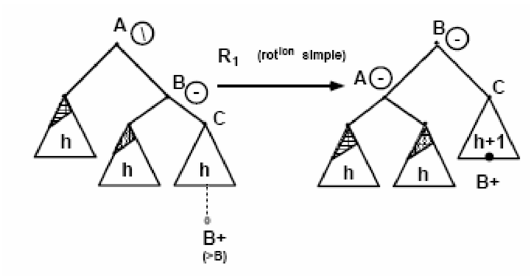
(KNUTH vol 3).

5. Arbres sous contrainte(s)

- Les insertions et les suppressions dans des arbres équilibrés en hauteur exigent une étude de cas
- Dans chaque noeud, un flag de balance valant / , - ou \ selon que le sous-arbre de gauche est >, = ou < au sous-arbre de droite. Au moment d'insertion (suppression), on est peut-être amené à réarranger l'arbre pour le maintenir équilibré en hauteur et le flag permet de déterminer quelles parties de l'arbre seront affectées

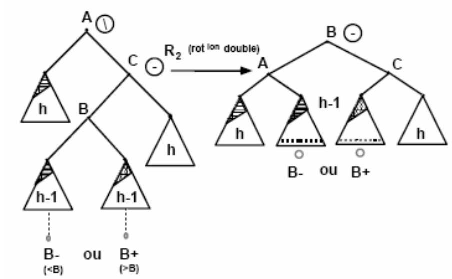
5. Arbres sous contrainte(s)

- Deux cas de base (+ cas symétriques) exigeant un réarrangement :



5. Arbres sous contrainte(s)

- Deux cas de base (+ cas symétriques) exigeant un réarrangement :



Ex : Fig. 6.18 et Fig. 6.19 (propagation).

5. Arbres sous contrainte(s)

b) Arbre équilibré en poids

- Un arbre complet correspond à un $\overline{t}_{rech}$ min $\Rightarrow$ à chaque insertion, restructurer l'arbre pour le maintenir complet ?

- Pour une possibilité de compromis entre MIN($\overline{t}_{rech}$) et la fréquence de réarrangements, voir NIEVERGELT et REINGOLD :
  - introduire un paramètre ρ qui permet de garder le déséquilibre entre les sous arbres de droite et gauche entre des limites raisonnables

5. Arbres sous contrainte(s)

- Soit un arbre binaire de N nœuds $T_N = <T_l, v, T_r>$:
  - $|T_N| = N = |T_l| + |T_r| + 1$

- Balance à la racine de T_N = $\rho(T_N) \triangleq \frac{|T_l| + 1}{N + 1}$

$$0 < \frac{1}{N+1} \leq \rho(T_N) \leq \frac{N}{N+1} < 1$$

- Définition - un arbre binaire T_N est dit de balance bornée α ou de type BB(α), $0 \leq \alpha \leq 1/2$, si :
  1. $\alpha \leq \rho(T) \leq 1 - \alpha$
  2. T_l et T_r sont de type BB(α)

5. Arbres sous contrainte(s)

- Pour les arbres complets remplis (complètement équilibrés) B_h, $N = 2^h-1$, $|T_l| = 2^{h-1}-1$ et $\rho(B_h) = 1/2$ $\Rightarrow$ les B_h sont BB(1/2)
- Les arbres de Fibonacci sont BB(1/3) et $\nexists$ d'arbre de type BB(α) tels que $1/3 < \alpha < 1/2$
- En effet, supposer $T \in$ BB(1/2) $\Rightarrow \exists$ sous-arbre de $T \notin$ BB(1/2)

5. Arbres sous contrainte(s)

- Soit T' le sous-arbre minimal de T qui $\in$ BB(1/2). T' est de la forme $<T'_l, v, T'_r>$ où T'_l et $T'_r \in$ BB(1/2) c'est-à-dire
  - $|T'_l| = 2^{l'} - 1$ et $|T'_r| = 2^{r'} - 1$ avec $r' \neq l'$

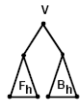
$$\rho(T') = \frac{2^{l'}}{2^{l'} + 2^{r'}} = \frac{1}{1 + 2^{r'-l'}} \leq \frac{1}{3} \text{ ou } \geq \frac{2}{3}$$

- selon que $r' < l'$ ou $r' > l'$ (ρ ou $1-\rho$)
  - {arbre équilibré en hauteur} $\not\subset \not\supset$ {arbre équilibré en poids}

5. Arbres sous contrainte(s)

- Exemple :
  - <B2,v,F4> $\in$ BB(13) mais pas équilibré en hauteur

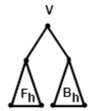
  - (Fh,v,Bh) équilibré en hauteur mais



$$\rho \simeq \frac{1}{1+\frac{\sqrt{5}}{4}\left(\frac{4}{1+\sqrt{5}}\right)^{h+2}}$$

r $\to$ 0 quand h $\to$ ∞ donc BB(0) c'est-à-dire non équilibré en poids

5. Arbres sous contrainte(s)



Équilibré en hauteur

$$|F_h| = f_{h+2} - 1 \simeq \frac{\Phi^{h+2}}{\sqrt{5}} = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2};$$

$$|B_h| = 2^h - 1$$

$$\rho = \frac{|F_h|+1}{|F_h|+|B_h|+2}$$

$$\simeq \frac{1}{1+\frac{\sqrt{5}}{4}2^{h+2}\left(\frac{2}{1+\sqrt{5}}\right)^{h+2}} \simeq \frac{1}{1+\frac{\sqrt{5}}{4}\left(\frac{4}{1+\sqrt{5}}\right)^{h+2}}$$

5. Arbres sous contrainte(s)

- Si T_N est BB(α) : $h_{max} \le \frac{\log(N+1)-1}{\log\left(\frac{1}{1-\alpha}\right)}$ * (cf réf. P. 222)
  - et $\bar{h} = E_p(N) = \frac{1}{H(\alpha)}(1+\frac{1}{N})\log(N+1)-2$
  - où $H(\alpha) = -\alpha\log\alpha-(1-\alpha)\log(1-\alpha)$

    $\alpha=\frac{1}{3} \Rightarrow \frac{1}{H(\alpha)} \simeq 1.09$
  - C'est-à-dire $\overline{t}_{rech}$ dans 1 BB(1/3) est au maximum 9% plus coûteux que dans 1 B_h de même N. De même, * $\Rightarrow$ t_{rech} max dans 1 BB(1/3) est au pire 70% plus coûteux que dans 1 B_h de même N

    $lb\ 3 = 1.59 \Rightarrow lb\frac{1}{1-\frac{1}{3}} = 0.59$ et $\frac{1}{lb\frac{3}{2}} \simeq 1.70$

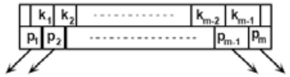
5. Arbres sous contrainte(s)

- Pour maintenir BB(α) :
  - mémoriser ρ à chaque noeud puis les mettre à jour à chaque insertion et suppression
  - réarrangement à chaque violation de la condition $\alpha \le \rho \le 1-\alpha$.
    - Si $\alpha \le 1-\frac{\sqrt{2}}{2}(0.2228)$, R_1 ou R_2 maintient BB(α).
    - Si les arbres sont supposés distribués $\rightarrow$ ρ est uniforme dans (α,1- α), le nombre de rotations/insertion < 2/(1-2α) est indépendant de N et pas prohibitif
- Exemple : Figure 6.21 (construire un arbre équilibré en poids)

  $\rightarrow$ insertions $\Rightarrow$ R_1 et R_2

5. Arbres sous contrainte(s)

c) Arbre lexicaux multi-voies



- à chaque noeud, (m-1) clés $k_1, ..., k_{m-1}$ et m pointeurs $p_1, ..., p_m$
- Quand une clé k est comparée à celles du noeud, p_j est sélectionné, tel que :

$$j = \begin{cases} 1 & \text{si } k \le k_1 \\ i & \text{si } k_{i-1} < k \le k_i \quad i = 2, ..., m-1 \\ m & \text{si } k_{m-1} < k \end{cases}$$

5. Arbres sous contrainte(s)

- Introduire des restrictions pour $\rightarrow$ sous-ensemble particuliers
  - B trees d'ordre m :
    - m/2 $\le$ nombre de pointeurs $\le$ m
    - Toutes les feuilles ont le même niveau
  - arb 2-3 $\in$ {B-trees} (m=3)
    - à chaque noeud, 1 ou 2 clés $\leftrightarrow$ 2 ou 3 pointeurs
    - toutes les feuilles au même niveau

5. Arbres sous contrainte(s)

- Soient N le nombre de clés et h la profondeur de l'arbre :

  $2^h - 1 \qquad 2.\sum_0^{h-1} 3^i = 2\frac{3^h-1}{3-1} = 3^h - 1$

- $2^h \leq N \leq 3^h$

  ⇒ $h \leq \log N \leq 1.59\, h$

  ⇒ $\overline{t}_{rech} = O(\log N)$.

- Construction identique à un arbre binaire lexicographique mais quand on a 3 clés dans un noeud, il y a éclatement et migration (centre) (avec propagation éventuelle). Exemple : figure 6.23

DS & PD in C++ 99 KRUSE & RYBA
p.535 External Searching : B-Trees
p.530 Lexicographic Search Trees : Tries

6. Arbres avec noeuds inégalement pondérés

- Soit un arbre avec uniquement des noeuds non vacants (c'est-à-dire où toute recherche est fructueuse), le nombre moyen de comparaisons est :

$$E_p(N) = \frac{1}{W}\sum_{i=1}^{N} p_i l_i \qquad \grave{a} \quad minimiser$$

- ⇔ ? choisir noeud le plus lourd comme racine ?

- Exemple : figure 6.24 :
  - cas (b) meilleur que cas (a) c'est-à-dire contre-exemple

6. Arbres avec noeuds inégalement pondérés

- En fait, si la séquence lexicographique $k_1<k_2<...<k_N$ a une séquence monotone de fréquence $p_1<p_2< ... <p_N$, on a une chaîne de longueur N

a) Arbre lexicographique binaire optimaux

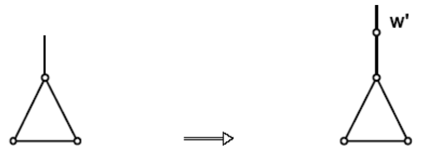
- Problème : Soient l'ensemble des clés lexicalement ordonnées $<k_1,...,k_N>$ de fréquence moyenne $<p_1,...,p_N>$ et de fréquence $<q_0,...,q_N>$ où q_i désigne la fréquence de recherche d'une clé se trouvant entre k_i et k_{i+1}. Construire l'arbre binaire minimisant

$$\sum_{j=1}^{N} p_j l_j + \sum_{k=0}^{N} q_k(l'_k - 1)$$

6. Arbres avec noeuds inégalement pondérés

- Figure 6.25 : $<q_0, ..., q_N>$ influe sur la forme de l'arbre optimisé
- Tout sous arbre d'un arbre optimal est optimal (sinon, le remplacement du sous arbre en question par un sous arbre optimal diminuerait la longueur des chemins).
- Figure 6.26 : construction par étapes de l'arbre lexicographique optimal : 1 puis 2 puis 3 puis 4 nœuds internes.
- Soient c(i,j) le coût (c'est-à-dire la longueur des chemins) de l'arbre optimal avec comme poids $(q_i,p_{i+1},q_{i+1},...,p_j,q_j)$ $0 \leq i \leq j \leq N$ et $w(i,j) \triangleq q_i+p_{i+1}+ ... +q_j$ la $\sum$ de ces poids.
- w(i,i) et c(i,i)=0 car aucun noeud interne

6. Arbres avec noeuds inégalement pondérés

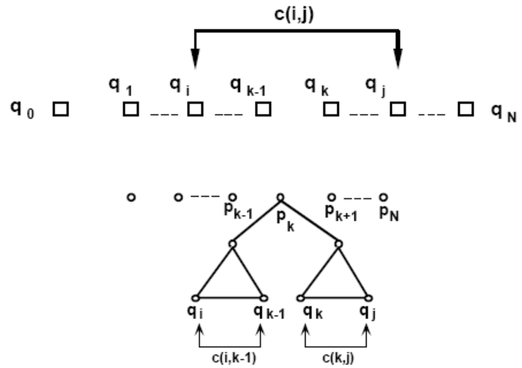


c(m,n)
= ∑ long chemins

c(m,n)+w(m,n)+w' car exact[t] les mêmes chemins avec chacun +1 longueur

optimal : $c(i,j) = w(i,j) + \min\limits_{i<k\leq l} \left[c(i,k-1) + c(k,j) \right] \quad i < j$

6. Arbres avec noeuds inégalement pondérés



6. Arbres avec noeuds inégalement pondérés

Exemple : Figure 6.21

w(i,j)

i \ j	0	1	2	3	4
0	0	10	18	24	28
1		0	12	18	22
2			0	9	13
3				0	6
4					0

c(i,j)

i \ j	0	1	2	3	4
0	0	10	28	43	57
1		0	12	27	40
2			0	9	19
3				0	6
4					0

$$c(i,j) = w(i,j) + \min_{i<k\leq j}[c(i,k-1) + c(k,j)] \quad i < j$$

6. Arbres avec noeuds inégalement pondérés

- à évaluer (j-i) fois pour chaque i allant de 1 à j et j va de 1 à N ⇒ nombre total d'opérations

$$\sum_{j=1}^{N}\sum_{i=1}^{j}(j-i) = \sum_{j=1}^{N}[j^2 - \frac{j(j+1)}{2}] \simeq \frac{N^3}{6}$$

- c'est-à-dire O(N3)
- Occupation de la mémoire pour [c(i,j)] et [w(i,j)] = $O(N^2)$
- KNUTH : ↘ $t_{exéc} = O(N^2)$

6. Arbres avec noeuds inégalement pondérés

- Arbres sub-optimaux
- $O(N^2)$ par bottom-up
- Si on a une construction top-down comme "noeud le plus lourd = racine" $\rightarrow$ $O(N \log N)$

  inadéquat entre autres parce que les q_i sont ignorées $\rightarrow$ "centroïde" $\triangleq$ racine minimisant la différence de poids entre les sous arbres de gauche et de droite

6. Arbres avec noeuds inégalement pondérés

Exemple

	A	B	C	D	E	F	G	H	I
$(p_i \mid q_i)w_i$	1	1	2	5	3	4	4	3	5
$\sum w_i$	1	2	4	9	12	16	20	23	28

centroïde

centroïdes pour 2e niveau

- "centroïde" marche pour les figures 6.24, 6.25(a) et 6.25(b)
- Soient C_{opt} le coût pour 1 arbre optimal et C_{cen} celui pour un arbre sub-optimal par "centroïde", BAYER : $\exists$ des bornes pour $C_{opt} \leq C_{cen}$

6. Arbres avec noeuds inégalement pondérés

- Amélioration en tenant compte des p_i individuels :

"centroïde" pur : $c(0,N)=125$

combinaison avec "noeud le plus lourd" : $c(0,N) = 97$

  - déterminer chaque fois le centroïde
  - dans son voisinage, trouver le maximum local
  - Ecart de l'optimal $\leq$ 3%

6. Arbres avec noeuds inégalement pondérés

- Programme pour construire 1 arbre sub-optimal à 2 paramètres :
  - F $\triangleq$ fraction de noeuds sur lesquels on recherche l'optimum local
  - s $\triangleq$ taille de la partition (nombre de noeuds) déclenchant la commutation vers l'algorithme optimal

6. Arbres avec noeuds inégalement pondérés

- Exemple
  - fichier de recensement canadien : 1/10^{e} $\rightarrow$ 1M de noms $\rightarrow$ 144486 noms distincts
  - N = 15 : on prend les 15 noms les plus courants (p_1, ..., p_{15} les plus élevées) $\rightarrow$ déduction des $q_0,\ldots,q_{15}$
    1. Quand la construction d'arbres optimaux et sub-optimaux est possible (jusque N=150), $\neq t_{rech} \leq 2\%$.
    2. Jusque N $\simeq$ 1000, $t_{rech} = O(\log N)$
    3. Quand N est élevé (>5000), $t_{rech} \simeq$ indépendant de N! car les derniers noms agrandissent l'arbre mais sont de moins en moins pondérés. Quand N = 144486, $q_i = 0\ \forall i$ et la longueur moyenne du chemin = 12.2, à comparer avec log 144486 - 1 $\simeq$ 16 d'un arbre à nœuds également pondérés

7. Arbres hybrides

- Noeuds généralisés :
  - soit des nœuds de caractères comme dans un trie
  - soit des noeuds de chaînes de caractères, pas nécessairement clés complètes
- Exemple : Figure 6.31

Références

- *Data structures and program design in C++* - KRUGS & RYBA, Prentice Hall 1999
  - Ch 10 – Binary trees- §10.4 : arbres équilibrés en hauteur (avl)
  - Ch 11 – Multiway trees- §11.2 : tries, §11.3 : B trees