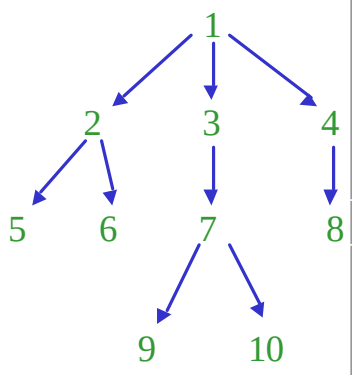
ARBRES

Arbre ordinaire : $A = (N,P)$
- N ensemble des nœuds
- P relation binaire « parent de »
- $r \in N$ la racine



$\forall x \in N \ \exists$ un seul chemin de r vers x

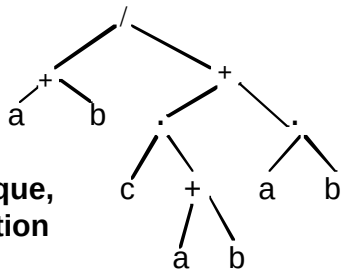
$r = y_0 \ P \ y_1 \ P \ y_2 \ \ldots \ P \ y_n = x$

$\Rightarrow r$ n'a pas de parent

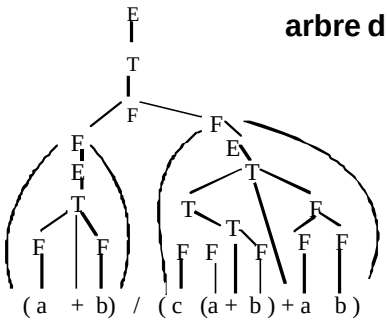
$\Rightarrow \forall x \in N - \{ r \}$ x a exactement un parent

Etiquettes

Étiquette : $N \rightarrow E$



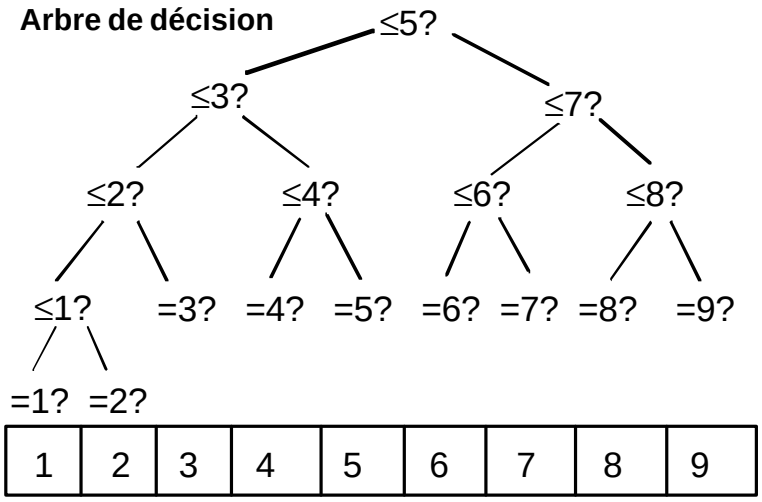
Arbre syntaxique, arbre d'exécution



Arbre d'analyse, pour une grammaire

UMLV©

Arbre de décision



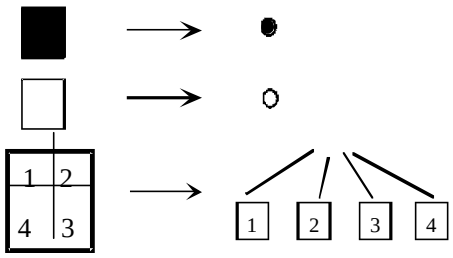
UMLV©

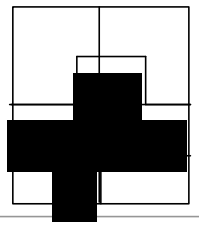
Arbre quartique

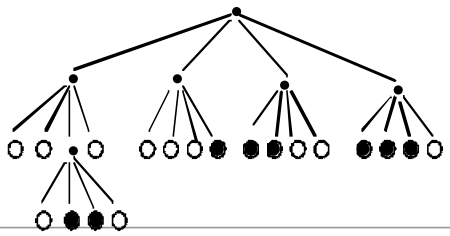
si image toute noire

si image toute blanche

sinon



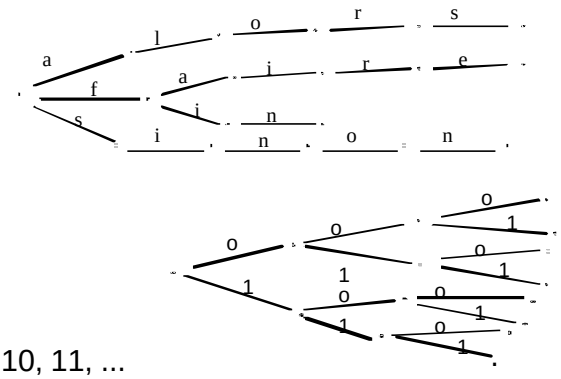




Arbres lexicographiques

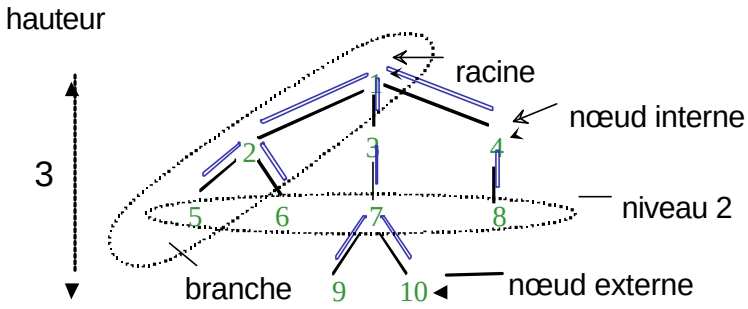
Arcs (*A*) = { (*x*,*y*) / *x* parent de *y* }

Étiquette : Arcs (*A*) → *E*



ε, 0, 1, 00, 01, 10, 11, ...
0 1 2 3 4 5 6 ...

Terminologie



2, 3, 4 enfants de 1
3, 4 frères de 2
1, 3, 7 ancêtres de 7
7, 9, 10 descendants de 7

Définitions récursives

UMLV©

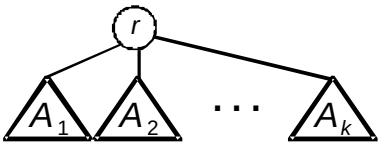
Arbre $A = \begin{cases} \Lambda & \text{arbre vide ou} \\ (r, \{A_1, ..., A_k\}) \end{cases}$

r élément, $A_1, ..., A_k$ arbres

Nœuds $(A) = \{r\} \cup$ ($\cup$ Nœuds (A_i))

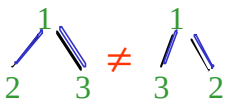
unions disjointes

$A = \Lambda$ ou



Arbre planaire $A = \begin{cases} \Lambda & \text{arbre vide ou} \\ (r, A_1, ..., A_k) \end{cases}$

Condition analogue



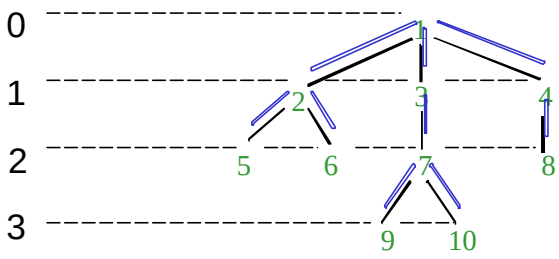
Niveaux

UMLV©

A arbre x nœud de A

$\text{niveau}_A(x)$ = distance de x à la racine

$\text{niveau}_A(x) = \begin{cases} 0 & \text{si } x = \text{racine}(A) \\ 1 + \text{niveau (parent } (x)\text{)} & \text{sinon} \end{cases}$



Hauteurs

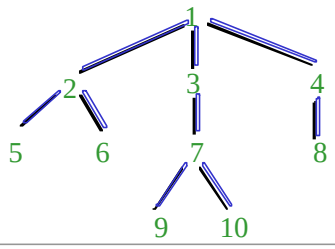
A arbre *x* nœud de *A*

$h_A(x)$ = distance de *x* à son plus lointain descendant qui est un nœud externe

$$h_A(x) = \begin{cases} 0 & \text{si } x \text{ nœud externe} \\ 1 + \max \{ h_A(e) \mid e \text{ enfant de } x \} & \text{sinon} \end{cases}$$

$h(A) = h_A(\text{racine}(A))$

$h_A(8) = 0$
$h_A(7) = 1$
$h_A(3) = 2$
$h(A) = h_A(1) = 3$



Sous-arbres

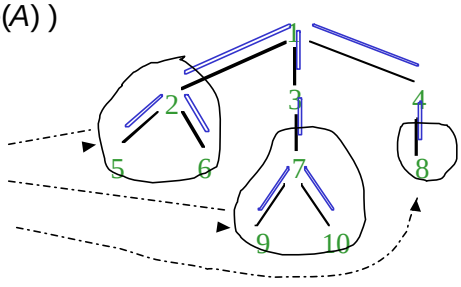
A arbre *x* nœud de *A*

$\text{Arbre}_A(x)$ = sous-arbre de *A* qui a racine *x*

$h_A(x) = h(\text{Arbre}_A(x))$

$A = \text{Arbre}_A(\text{racine}(A))$

$\text{Arbre}_A(2)$
$\text{Arbre}_A(7)$
$\text{Arbre}_A(8)$



Parcours

UMLV©

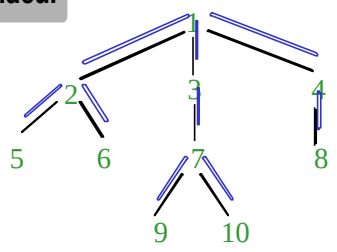
Fonction : arbre → liste de ses nœuds
arbre vide → liste vide

Utile pour l 'exploration des arbres

Deux types :
- parcours en profondeur
  - préfixe, suffixe, symétrique
  - parcours branche après branche
- parcours en largeur ou hiérarchique
  - parcours niveau après niveau

Parcours en profondeur

UMLV©



Arbre non vide $A = (r, A_1, A_2, \ldots, A_k)$

Parcours préfixe
$P(A) = (r).P(A_1). \ldots .P(A_k)$ (1, 2, 5, 6, 3, 7, 9, 10, 4, 8)

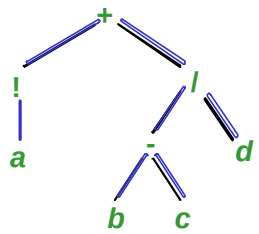
Parcours suffixe
$S(A) = S(A_1). \ldots .S(A_k).(r)$ (5, 6, 2, 9, 10, 7, 3, 8, 4, 1)

Parcours symétrique (ou interne)
$I(A) = I(A_1).(r).I(A_2). \ldots .I(A_k)$ (5, 2, 6, 1, 9, 7, 10, 3, 8, 4)

Expressions arithmétiques

UMLV©

Arbre syntaxique de $a! + \frac{b-c}{d}$



Parcours préfixe

+ ! a / - b c d

Parcours suffixe

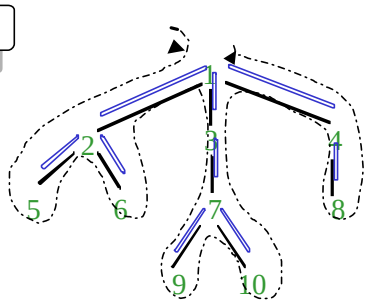
a ! b c - d / +

Parcours symétrique (priorités et parenthèses)

(a !) + ((b - c) / d)

Rencontres

UMLV©



Parcours préfixe = première rencontre

(1, 2, 5, 6, 3, 7, 9, 10, 4, 8)

Parcours Suffixe = dernière rencontre

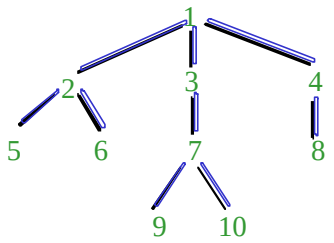
(5, 6, 2, 9, 10, 7, 3, 8, 4, 1)

Parcours Symétrique = deuxième rencontre

(5, 2, 6, 1, 9, 7, 10, 3, 8, 4)

Parcours en largeur

UMLV©



Arbre non vide $A = (r, A_1, A_2, \ldots, A_k)$

Parcours hiérarchique

$H(A) = (r, x_1, \ldots, x_i, x_{i+1}, \ldots x_j, x_{j+1}, \ldots, x_n)$

nœuds de niveau 0, 1, 2, ...

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

Type arbre

UMLV©

Ensemble

Arbres planaires de nœuds étiquetés

Opérations

Arbre-vide : ® arbre
Racine : arbre ® nœud
Enfants : arbre ® liste d'arbres
Cons : nœud x liste d'arbres ® arbre
Vide : arbre ® booléen
Elt : nœud ® élément

Axiomes principaux

Racine(*A*) et Enfants(*A*) définis ssi *A* non vide
Racine (Cons(*r, L*)) = *r*
Enfants(Cons(*r, L*)) = *L*

Implémentation

UMLV©

Représentation de la relation *P*
table des parents

Avantages
représentation simple
parcours faciles vers la racine
économique en mémoire

Inconvénients
accès difficiles aux nœuds depuis la racine

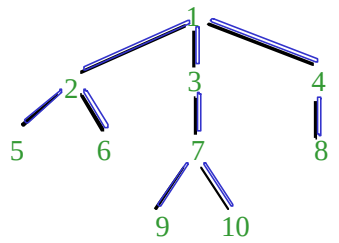


table des parents *P*

-	1	1	1	2	2	3	4	7	7
1	2	3	4	5	6	7	8	9	10

Implémentation (suite)

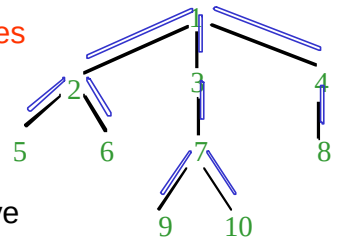
UMLV©

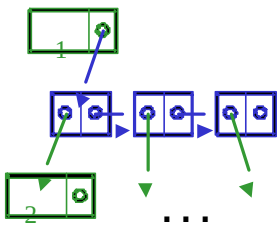
Représentation des listes de sous-arbres par chaînage

Avantages
accès faciles depuis la racine
correspond à la définition récursive

Inconvénients
parcours difficiles vers la racine
relativement gourmand en mémoire

Deux types de pointeurs
Arbre souvent identifié à l 'adresse de sa racine (comme pour un tableau en C)





Fonction Préfixe

UMLV©

```
fonction Préfixe (A arbre) : liste de nœuds ;
début
        si A = arbre vide alors retour (suite vide)
        sinon {
                L ¬ ( Racine (A) ) ;
                pour B ¬ premier au dernier élément de Enfants(A) faire
                        L ¬ L . Préfixe(B) ;
                retour ( L ) ;
        }
fin
```

Temps d 'exécution : $O(n)$
sur un arbre à n nœuds représenté par pointeurs

Fonction suffixe

UMLV©

```
fonction Suffixe (A arbre) : liste de nœuds ;
début
        si A = arbre vide alors retour (suite vide)
        sinon {
                L ¬ ( ) ;
                pour B ¬ premier au dernier élément de Enfants(A) faire
                        L ¬ L . Suffixe(B) ;
                L ¬ L . (racine(A) ) ;
                retour ( L ) ;
        }
fin
```

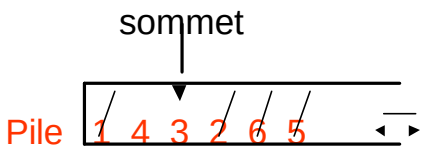
Temps d 'exécution : $O(n)$
sur un arbre à n nœuds représenté par pointeurs

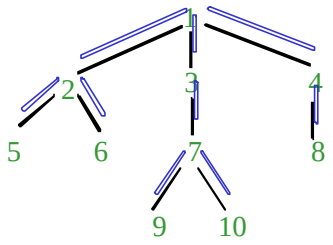
Fonction Préfixe itérative


```
fonction Préfixe (A arbre) : liste de nœuds ;
début
        L ¬ ( ) ;
        Pile ¬ Empiler( Pile-vide, A ) ;
        tant que non Vide( Pile ) faire {
                A' ¬ sommet ( Pile ) ;   Pile ¬ Dépiler ( Pile ) ;
                si A' non vide alors {
                        L ¬ L . (racine(A') ) ;
                        pour B ¬ dernier au premier élément de Enfants(A') faire
                                Pile ¬ Empiler ( Pile, B ) ;
                }
        }
        retour ( L ) ;
fin
```

Temps d'exécution $O(n)$ comme version récursive
si, en plus, bonne implémentation de la pile

Exemple





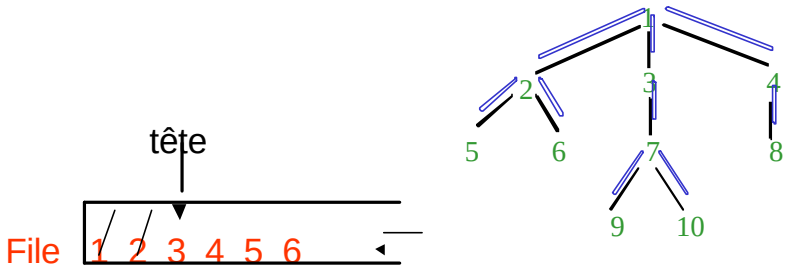
Liste L = (1, 2, 5, 6, .

Fonction Niveaux


```
fonction Niveaux (A arbre) : liste de nœuds ;
début
        L ¬ ( ) ;
        File ¬ Enfiler( File-vide, A ) ;
        tant que non Vide( File ) faire {
                A' ¬ tête ( File ) ;  File ¬ Enlever ( File ) ;
                si A' non vide alors {
                        L ¬ L . (racine(A') ) ;
                        pour B ¬ premier au dernier élément de Enfants(A') faire
                                File ¬ Ajouter ( File, B ) ;
                }
        }
        retour ( L ) ;
fin
```

Temps d'exécution $O(n)$ si bonne implémentation de la file pas de version récursive

Exemple



Liste L = (1, 2, .

Arbres k-aires

UMLV©

Arbre k-aire : tout nœud possède k sous-arbres (vides ou non), k fixé

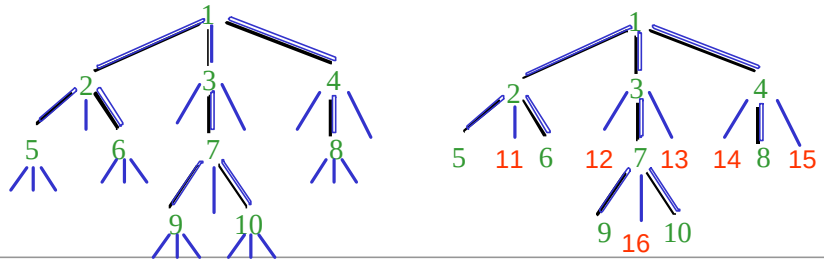
$A = \begin{cases} \Lambda & \text{arbre vide ou} \\ (r, A_1, ..., A_k) & \end{cases}$

r élément, $A_1, ..., A_k$ arbres k-aires

Nœuds $(A) = \{r\} \cup (\cup$ Nœuds $(A_i))$

unions disjointes

Arbre k-aire complet : tout nœud interne possède k enfants



Arbres binaires

UMLV©

Arbre binaire : tout nœud possède deux sous-arbres (vides ou non)

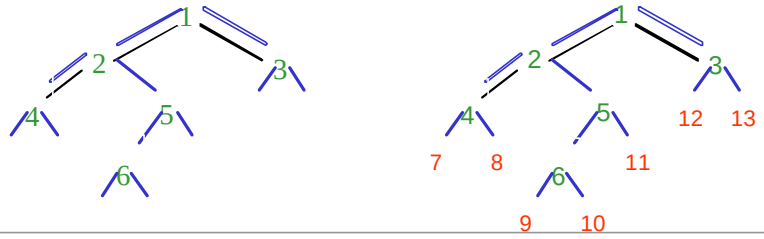
$A = \begin{cases} \Lambda & \text{arbre vide ou} \\ (r, G, D) & \end{cases}$

r élément, G, D arbres binaires

Nœuds $(A) = \{r\} \cup$ Nœuds $(G) \cup$ Nœuds (D)

unions disjointes

Arbre binaire complet : tout nœud interne possède deux enfants



Type arbre binaire

UMLV©

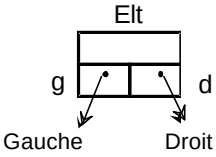
Ensemble

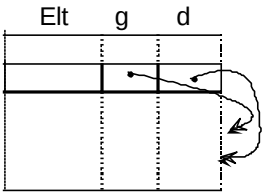
Arbres binaires étiquetés

Opérations

Arbre-vide : ® arbre
Racine : arbre ® nœud
Gauche, Droit : arbre ® arbre
Cons : nœud x arbre x arbre ® arbre
Elt : nœud ® élément
Vide : arbre ® booléen

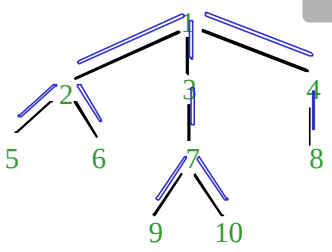
Implémentations par pointeurs ou curseurs

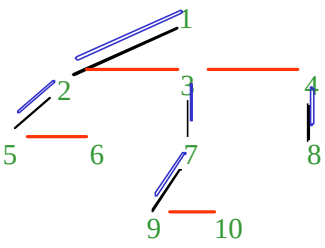




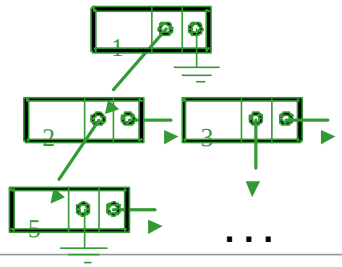
Binarisation

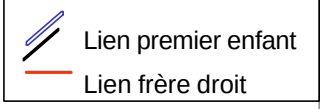
UMLV©





Implémentation des arbres planaires





Un seul type de pointeur

Arbres feuillus

UMLV©

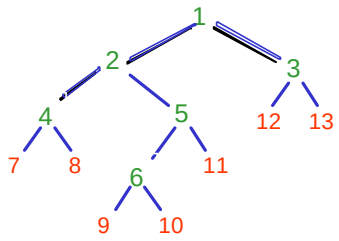
Arbre binaire feuillus (complets) : deux types de nœuds, internes ou feuilles
-- tout nœud interne possède deux enfants ;
-- toute feuille est un nœud externe.

$A = \begin{cases} (f) & f \text{ de type feuille} \\ (r, G, D) & r \text{ de type interne , } G, D \text{ arbre binaires feuillus} \end{cases}$

Nœuds (A) = {r} ∪ Nœuds (G) ∪ Nœuds (D) unions disjointes

Nœuds internes : 1, 2, 3, 4, 5, 6

Feuilles : 7, 8, 9, 10, 11, 12, 13



Taille des arbres feuillus

UMLV©

Arbre feuillu : nombre de feuilles = nombre de nœuds internes + 1

Récurrence sur le nombre de nœuds de l 'arbre A :

- si un seul nœud, c 'est une feuille ; propriété satisfaite.

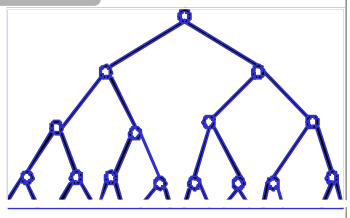
- sinon, il existe un nœud interne dont les enfants sont des feuilles, *i.e.* un sous-arbre (x, g, d) où g, d sont des feuilles.
Soit B obtenu de A en remplaçant (x, g, d) par une feuille f.
B est un arbre feuillu de plus petite taille ; par récurrence l 'égalité est satisfaite sur B ; donc aussi sur A qui possède un nœud interne et une feuille de plus. CQFD.

Arbre feuillu : $2.m + 1$ nœuds

Mesures des arbres binaires

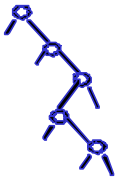
UMLV©

Arbre binaire, hauteur h et n nœuds



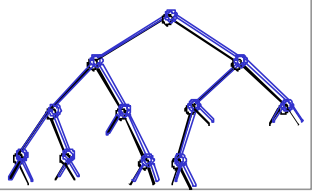
Arbre plein

2^i nœuds au niveau i
$n = 2^{h+1} - 1$



Arbre filiforme

1 nœud par niveau
$n = h + 1$



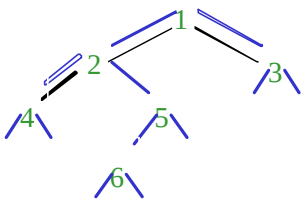
Arbre binaire

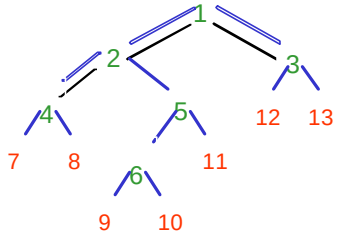
$h + 1 \leq n \leq 2^{h+1} - 1$
$\log_2 (n + 1) - 1 \leq h \leq n - 1$

Bijection

UMLV©

Arbres binaires à n nœuds « arbres binaires complets à $2.n+1$ nœuds

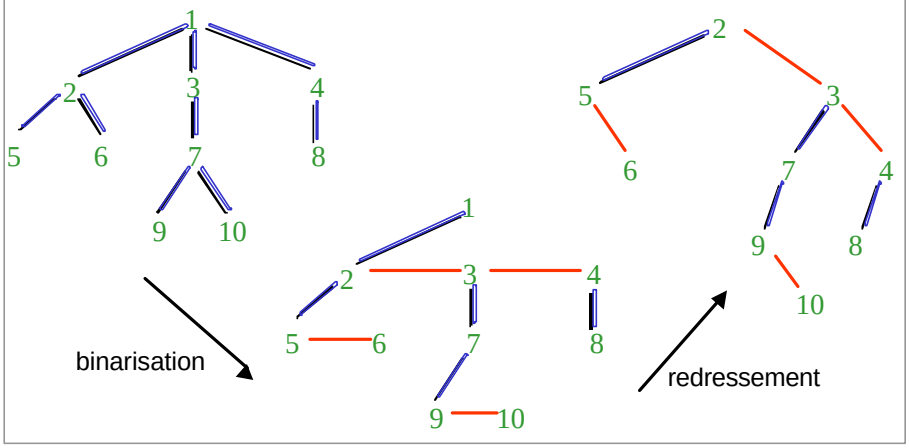




Autre bijection

UMLV©

Arbres planaires à n+1 nœuds « arbres binaires à n nœuds

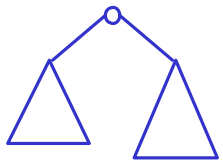


Énumération

UMLV©

C_n = nombre d'arbres binaires complets à n feuilles

$$\begin{cases} C_0 = 0, \quad C_1 = 1 \\ C_n = \sum_{i=1}^{n-1} C_i C_{n-i} \quad n \geq 2 \end{cases}$$



i feuilles n-i feuilles

Série génératrice : $s(t) = \sum_{n \geq 1} C_n t^n$

$$s(t) = s(t)^2 + t \quad \Rightarrow \quad s(t) = \frac{1 - \sqrt{1-4t}}{2}$$

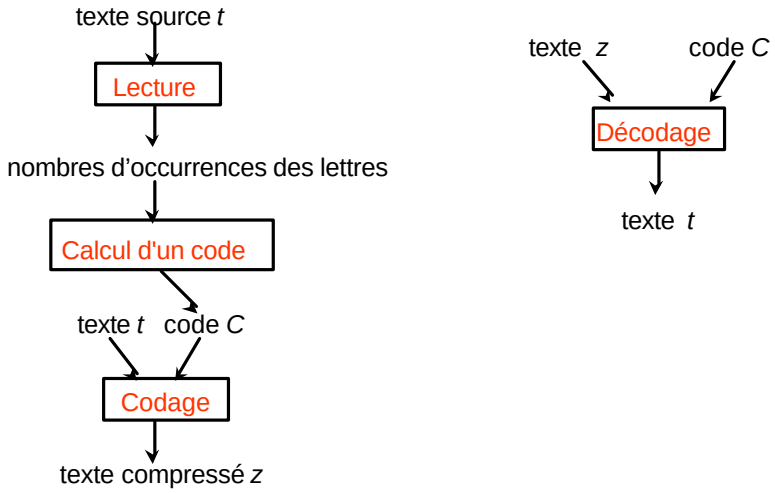
$$\sqrt{1-4t} = \sum_{n \geq 0} (-1)^n 4^n t^n \frac{1/2 \ (1/2\text{-}1)\ldots(1/2 - n + 1)}{n!}$$

$$\Rightarrow \quad C_n = \frac{1}{2n-1}\binom{2n-1}{n} = \frac{(2n-2)!}{n!(n-1)!}$$

nombres de catalan

Compression

UMLV©



Codes Préfixes

UMLV©

$C \subseteq \{0,1\}^*$

C code préfixe ssi $u,v \in C$ et *u* préfixe de $v \Rightarrow u = v$

aucun mot de *C* n'est un préfixe d 'un autre mot de *C*

{ 00,01,10,11 }	{ 0,10,11 }
a b c d	a b c
code ASCII	0*1

décodage unique et séquentiel de $x \in C^*$ (avec $C = 0^*1$)

0 0 1 | 1 | 0 0 0 0 1 | 0 1 | 1 | 1 | 0 0 1 | ...

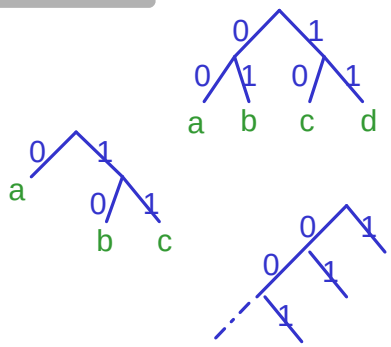
Bijection

UMLV©

{ 00,01,10,11 }
a b c d

{ 0,10,11 }
a b c

0*1



Code préfixe ⟷ Arbre binaire

mot du code — nœud externe

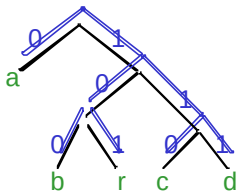
Compression statistique

UMLV©

Codage des caractères en fonction de leur fréquence

t = a b r a c a d a b r a

a	5 fois	0
b	2 fois	1 0 0
c	1 fois	1 1 0
d	1 fois	1 1 1
r	2 fois	1 0 1



Texte codé
z = 0 1 0 0 1 0 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 1 0
23 bits (si on ne compte pas le code) contre 11 x 8 en ASCII

Décodage
0 | 1 0 0 | 1 0 1 | 0 | 1 1 0 | 0 | 1 1 1 | 0 | 1 0 0 | 1 0 1 | 0 |
a b r a c a d a b r a

Arbres pondérés

UMLV©

t texte initial, *z* texte compressé
p(*a*) = nombre d'occurrences de *a* dans *t*
h(*a*) = mot du code associé à *a*

$$|z| = \sum_{a \in A} p(a).|h(a)|$$

Problème :

connaissant $p : A \longrightarrow \mathbb{N} - \{0\}$

calculer $h : A \longrightarrow \{0,1\}^*$ tel que - *h*(A) est un code préfixe et
- $\Sigma\ p(a).|h(a)|$ minimal

Problème équivalent :

déterminer un *A* arbre binaire feuillu (complet) dont les feuilles sont les lettres de A, avec p : Feuilles(*A*) $\longrightarrow \mathbb{N} - \{0\}$, tel que

Poids(*A*) = $\Sigma\ p(f).\text{niveau}(f)$ est minimal
f feuille de *A*

Arbres de Huffman

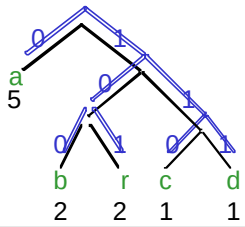
UMLV©

Arbre pondéré :

A arbre binaire feuillu (complet) dont les feuilles sont les lettres de A
p : Feuilles(*A*) $\longrightarrow \mathbb{N} - \{0\}$

Poids(*A*) = $\Sigma\ p(f).\text{niveau}(f)$
f feuille de *A*

A arbre de Huffman (pour *p*) si Poids(*A*) minimal

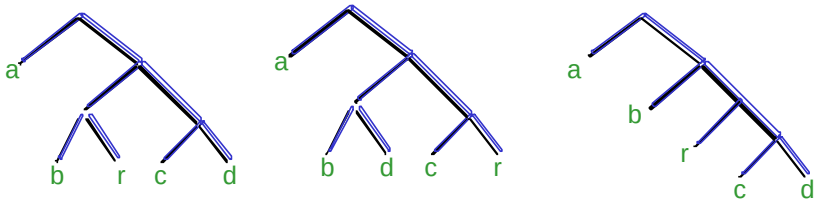


Poids(*A*) = longueur du texte compressé =
5 x 1 + 2 x 3 + 2 x 3 + 1 x 3 + 1 x 3 = 23

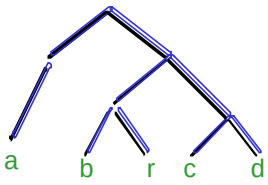
Exemple (suite)

UMLV©

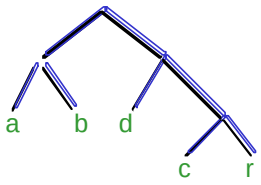
Trois arbres de Huffman (Poids = 23)



Poids = 28



Poids = 25



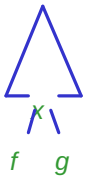
Propriétés

UMLV©

A arbre pondéré pour p : A $\longrightarrow \mathbb{N}$ - {0} ; A = Feuilles(*A*)

1 - *A* arbre de Huffman $\Rightarrow$ *A* arbre binaire complet

2 - *A* arbre de Huffman et card A > 1
Alors, à une permutation des feuilles près,
A possède un sous-arbre (*x*, *f*, *g*) où *f*, *g* sont des feuilles telles que $p(f)$, $p(g)$ minimaux



3 - Soit *B* obtenu de *A* en y remplaçant (*x*, *f*, *g*) par la feuille *y* ;
soit *q* : Feuilles(*B*) $\longrightarrow \mathbb{N}$ - {0} définie par

$$\begin{cases} q(y) = p(f) + p(g) \\ q(u) = p(u) \quad \text{pour } u \neq y \end{cases}$$

Alors, *A* arbre de Huffman pour *p* ssi *B* arbre de Huffman pour *q*

Preuve : car Poids(*A*) = Poids(*B*) + $p(f)$ + $p(g)$

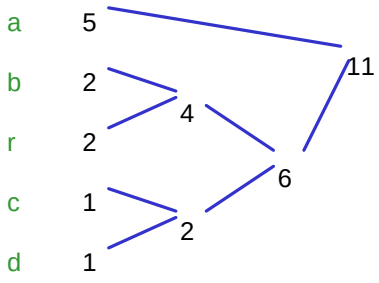
Algorithme de Huffman

$p : A \longrightarrow \mathbb{N} - \{0\}$

arbre élémentaire $X_a = (x_a)$, pour chaque lettre $a \in A$

fonction Huffman (alphabet A, fonction *p*)
début
    $L \leftarrow (X_a \mid a \in A)$;
    tant que $|L| > 1$ **faire {**
        B, C ← arbres de *L* avec *p*(*B*), *p*(*C*) minimaux ;
        Y ← (nouveau-nœud, *B*, *C*) ; *p*(*Y*) ← *p*(*B*) + *p*(*C*) ;
        remplacer *B* et *C* dans *L* par *Y* ;
    }
    retour l'unique arbre de *L* ;
fin

Exemple



Arbre de Huffman

a	0
b	1 0 0
r	1 0 1
c	1 1 0
d	1 1 1

Code de Huffman

Implémentation

		p	*g*	*d*
1	a	5	—	—
2	b	2	—	—
3	r	2	—	—
4	c	1	—	—
5	d	1	—	—
6	—	2	4	5
7	—	4	2	3
8	—	6	7	6
9	—	11	1	8

Table de taille 2.card A - 1

Calcul : tri en temps *O*(card A x log card A)
construction de l 'arbre en *O*(card A)