Stratégie de compression sans perte : contraintes et méthodes

Codage optimal de symboles, sans préfixes

Arbre pondéré de Huffman et entropie

Optimisation : prétraitements

Optimisation : représentation compacte d'arbre canonique

Limitation de la longueur des codes : de Huffman à Package-Merge

Notion de compression sans perte

Données à compresser : représentation dans un code C1 d'une suite d'éléments (ou symboles) d'un alphabet.

Données compressées : représentation plus compacte dans un code C2 de la même suite d'éléments.

Sans perte d'information : opération de compression réversible.

Efficacité et choix de C2 : selon redondance détectée dans le codage des données en C1.

Contextes et contraintes d'utilisation

On se limite aux méthodes représentant les informations par des codes à nombre entier de bits (exclut les méthodes quadratiques).

Approche **statique** applicable uniquement aux **fichiers** : le code C2 est défini statiquement **après** analyse de l'ensemble du contenu à compresser

$\rightarrow$ ce que l'on vous demande.

Approche **dynamique/adaptative** pour les flux, sans parcours préalable des données à compresser. Le code C2 ne dépend que de la partie du contenu déjà traitée et évolue avec elle.

$\rightarrow$ une extension possible du travail pour les forts/curieux.

Stratégies de compression sans perte

Optimisation du codage de séquences : tous les codes même longueur, un code par symbole ou séquence de symboles.

- suites de symboles identiques : RLE
- codage de sous-séquences déjà vues : Lampel-Ziv (ricm3 2015)

Optimisation du codage symbole par symbole : code de longueur variable selon la fréquence d'apparition du symbole. Le codage repose sur un arbre binaire (à décrire dans le contenu compressé).

- **Huffman** : arbre optimal hauteur/longueur de code max = n-1
- Arbre et codes de longueur bornée : algorithme **Package-Merge**

$\rightarrow$ le thème de cette année

Codage individuel de symboles

Cas simple : symbole $\simeq$ octet de donnée à compresser

Alphabet de symboles typique :

1. 128 (texte ASCII) ou 256 (binaire/UTF/ISO) valeurs possibles d'octet
2. marqueur de fin de données : End Of File
3. gestion interne de (autre) compression : (distance,longueur) de LZ77.

Principe de l'optimisation : le nombre de bits du code d'un symbole est adapté à la fréquence d'apparition du symbole.

Codage décodable

Suite de symboles à transcoder = une suite (de séquences de) bits codant les symboles.

Taille réduite : absence de marque sépciale de séparation de symboles

Découpage correct de la suite de bits possible
$\rightarrow$ pas de code préfixe d'un autre code.

Contre exemple : 00 (a) préfixe de 0 (b) et 11 (c) préfixe de 1 (d).
Comment interprétér 0011 ?

1. ac
2. bbdd

Arbre binaire pour codes sans préfixes

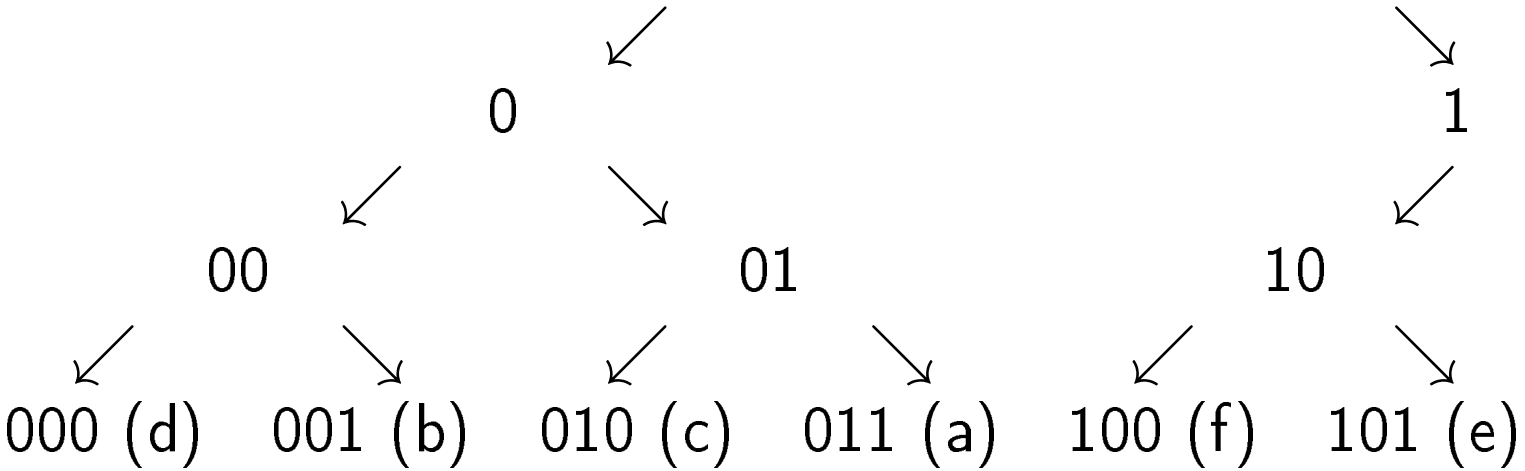
- Code = parcours d'arbre binaire depuis la racine (0 = gauche, 1 = droite)
- Nœuds sans fils = feuilles à code sans préfixe : y placer les symboles
- Nœuds avec fils : code préfixe des codes des feuilles des sous-arbres gauche et droit : aucun symbole

1. Arbre "aplati" : hauteur minimale (selon $log_2(nombresymboles)$) identique pour toutes les feuilles symboles
2. Codage en longeur variable : forme d'arbre optimale selon nombre d'occurences des symboles.

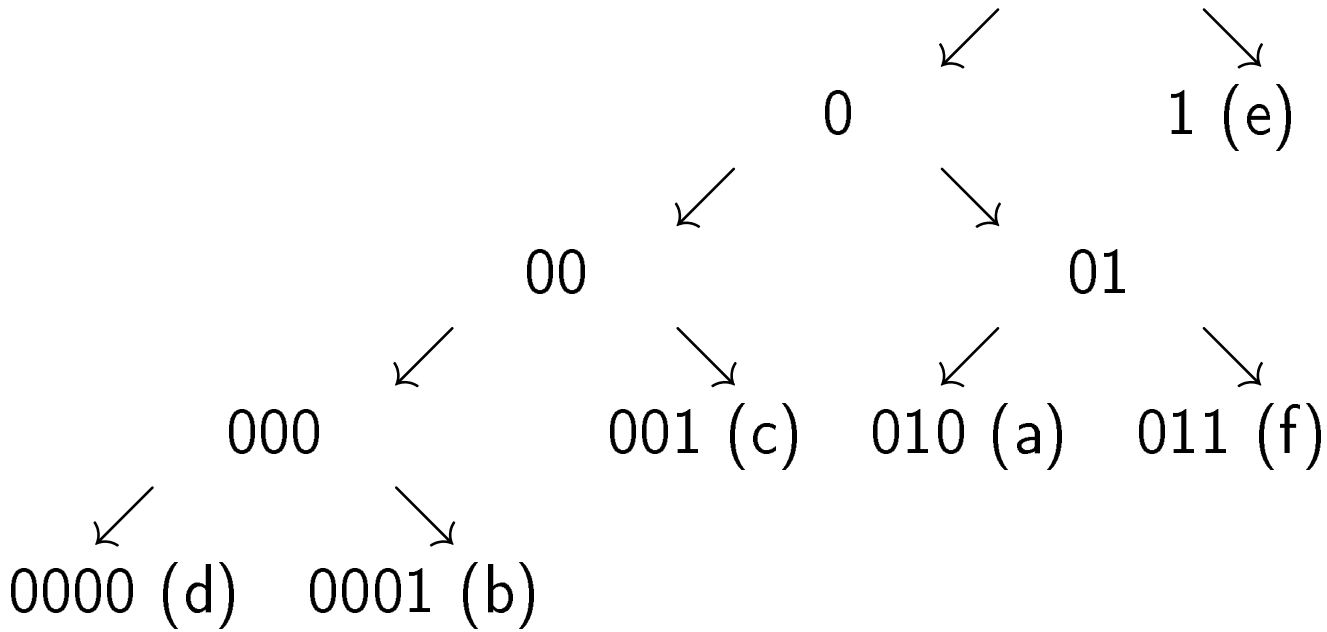
Exemple : distribution des occurences

symbole	occ	code1	long1	bits	code2	long2	bits
a	5	011	3	15	010	3	15
b	3	001	3	9	0001	**4**	**12**
c	4	010	3	12	001	3	12
d	2	000	3	6	0000	**4**	**8**
e	18	101	3	54	1	**1**	**18**
f	7	100	3	21	011	3	21
total	39			117			86

Codage 1 de l'exemple : arbre "aplati"



Codage 2 de l'exemple optimisé selon les occurences



Interprétation de codes en taille variable

Les données sont codées comme une suite de bits. L'unité élémentaire de taille et d'accès aux fichiers et à la mémoire centrale est l'octet.

Comment grouper les bits en codes de symboles (à la décompression) ?

Comment savoir quand on a traité le dernier bit de contenu compressé à décoder ?

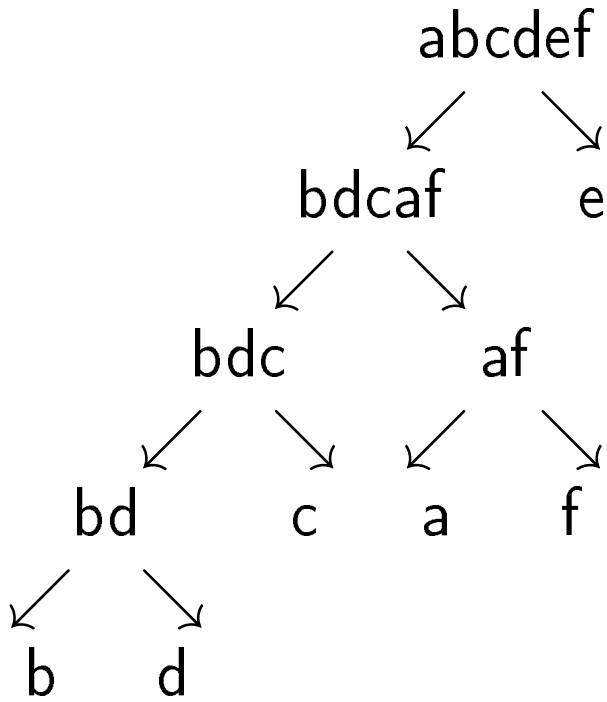
- Aucun code de symbole dans le fichier compressé ne doit être le préfixe (début de séquence de bits) du code d'un autre symbole.
- L'en-tête du fichier compressé donne le nombre de symboles ou on ajoute au flux de données à compresser un symbole marque de fin de données.

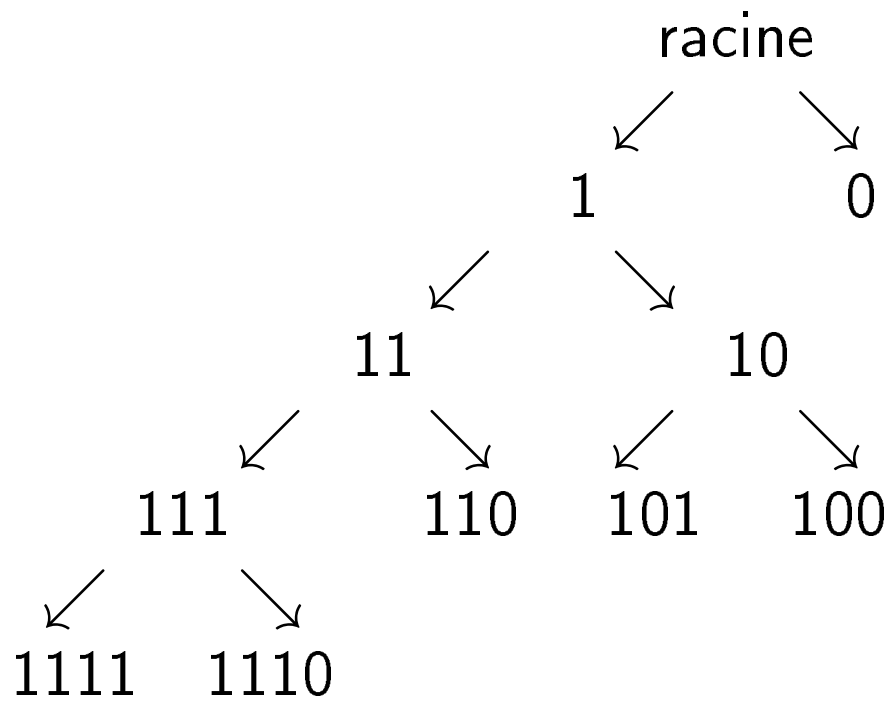
Retour : sans préfixe = arbre binaire

Codage "préfixe-free" $\rightarrow$ description par un arbre binaire :

- feuilles : symboles.
- nœuds non feuilles : groupements de symboles
  - fils gauche : sous-arbre étiqueté 1 non vide.
  - fils droit : sous-arbre étiqueté 0 non vide.
- code de symbole : parcourir étiquettes (0,1) depuis la racine.
- par construction : aucune feuille ascendant d'une feuille
  $\rightarrow$ aucun code de symbole préfixe du code d'un autre symbole.
- choix : groupements tassés à gauche, feuilles tassées à droite.
- taille du code d'un symbole : profondeur de la feuille.

Exemple d'arbre et de codage binaires





Exemple d'alphabet de 6 symboles : {a,b,c,d,e,f}

b : 1111, d : 1110 c : 110, a : 101, f : 100, e : 0

fréquences croissantes = longueurs décroissantes $\longrightarrow$

Entropie de Shannon et quantité d'information

Entropie : quantité d'information $H = -(p_0 log_2(p_0) + p_1 log_2(p_1))$

Deux définitions différentes de bit :

1. Binary digIT : chiffre de la base 2.
2. Unité élémentaire d'information.

Définitions équivalentes si valeurs 0 et 1 équiprobables.

L'information dans un bit (1) est maximale lorsque les probabilités d'apparition de 0 et 1 sont égales : 1 bit (2).

- jet pile/face de pièce non truquée : imprévisible.
  $H = -(\frac{1}{2} log_2(\frac{1}{2}) + \frac{1}{2} log_2(\frac{1}{2})) = 1 bit$
- jour = 29 février prévisible $\simeq \frac{1}{4*365+1} = 0.068\%$ des cas.
  $H \simeq -(\frac{1}{1461} log_2(\frac{1}{1461}) + \frac{1460}{1461} log_2(\frac{1460}{1461})) \simeq 0.082 bit$

Arbre pondéré

Pondération des nœuds par les fréquences cumulées :

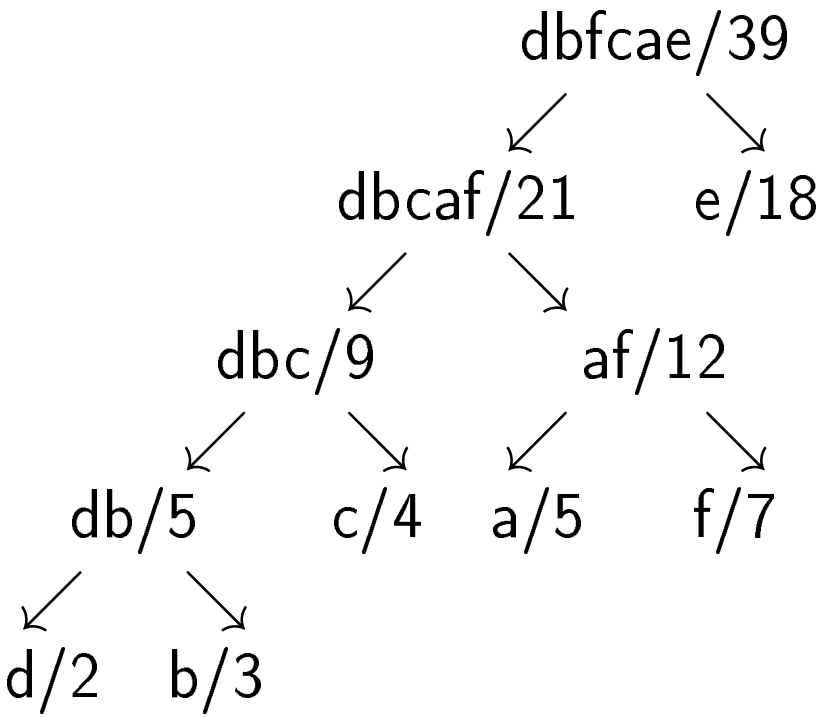
- Le poids d'une feuille est la fréquence d'apparition du symbole feuille.
- Le poids d'un (nœud racine d'un) sous-arbre est la somme des poids de ses sous-arbre gauche et droit.

Soit h_i la prodondeur (= longueur du code) du symbole i et f_i son poids, le nombre de bits de la représentation compressée des symboles (hors tables des longueurs de codes) est :

$$\textit{Coût de codage} = \sum_{i \in \textit{noeuds}} f_i = \sum_{i \in \textit{feuilles}} h_i f_i$$

Arbre pondéré : objectifs

- maximiser la quantité d'information contenue dans chaque bit
- chaque bit du contenu compressé permet de choisir entre 2 sous-arbres
- équilibrage des poids gauche et droit en chaque nœud



Arbre (non canonique) pour l'exemple à 39 symboles (2 fois d, 3 fois b, 4 fois c, 5 fois a, 7 fois f, 18 fois e).

Gain min et max

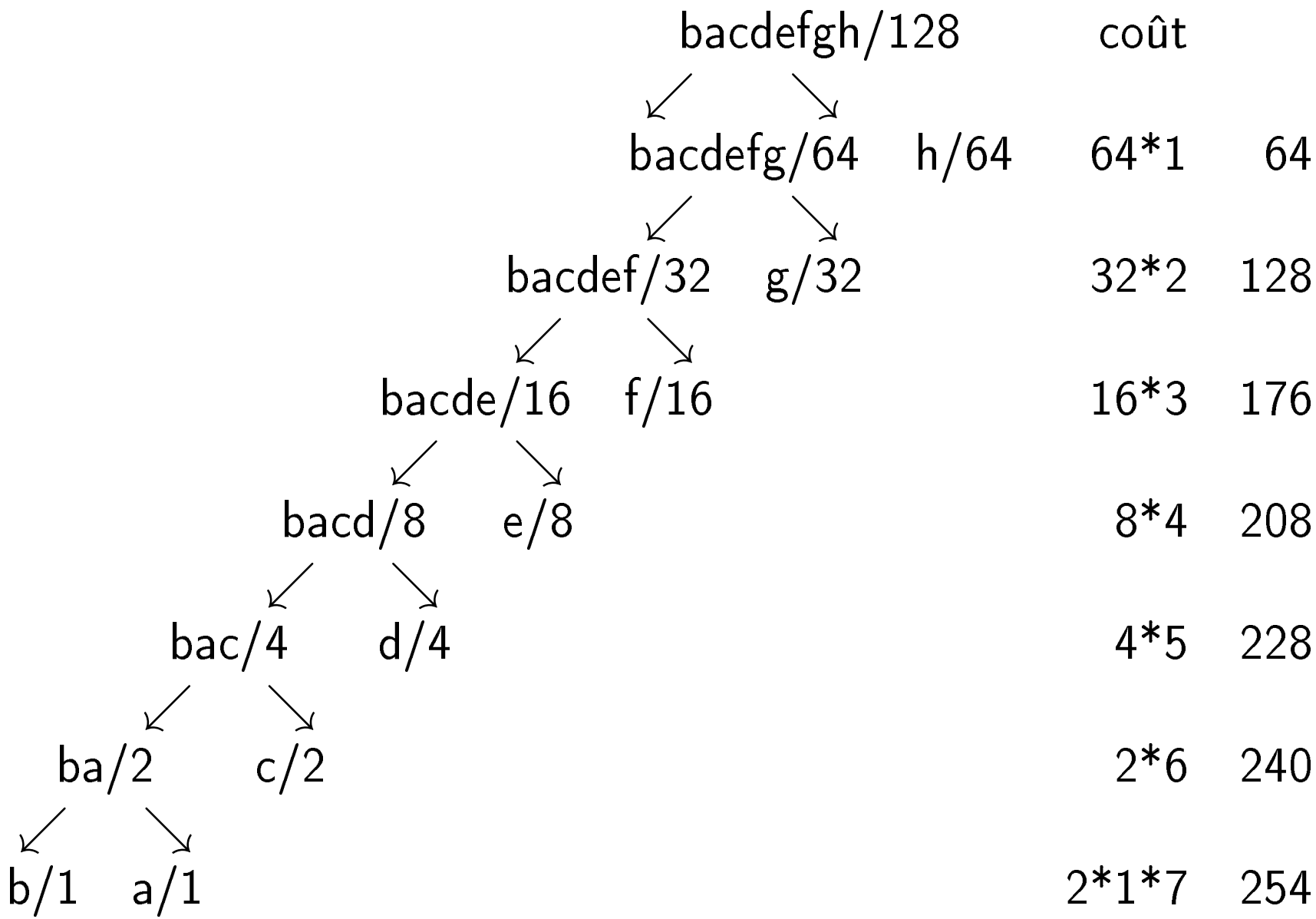
Fréquences homogènes $\rightarrow$ graphe équilibré $\rightarrow$ gain faible.

Cas limite négatif : $N = 2^n$ et $\forall s, \mathit{frequence}[s] = f$.

- graphe équilibré et complet
- codage initial en taille fixe déjà optimal : gain nul.

Cas limite positif : fréquences en 2^i, 128 codes de 8 symboles.

- codage en taille fixe : 3*128 = 384 bits
- codage en taille variable : 254 bits (-34%)


bacdefgh/128
coût
bacdefg/64
h/64
64*1
64
bacdef/32
g/32
32*2
128
bacde/16
f/16
16*3
176
bacd/8
e/8
8*4
208
bac/4
d/4
4*5
228
ba/2
c/2
2*6
240
b/1
a/1
2*1*7
254

Algorithme de Huffman

Règle de choix des paires à grouper : grouper en priorité les sous-arbres de plus faibles poids.

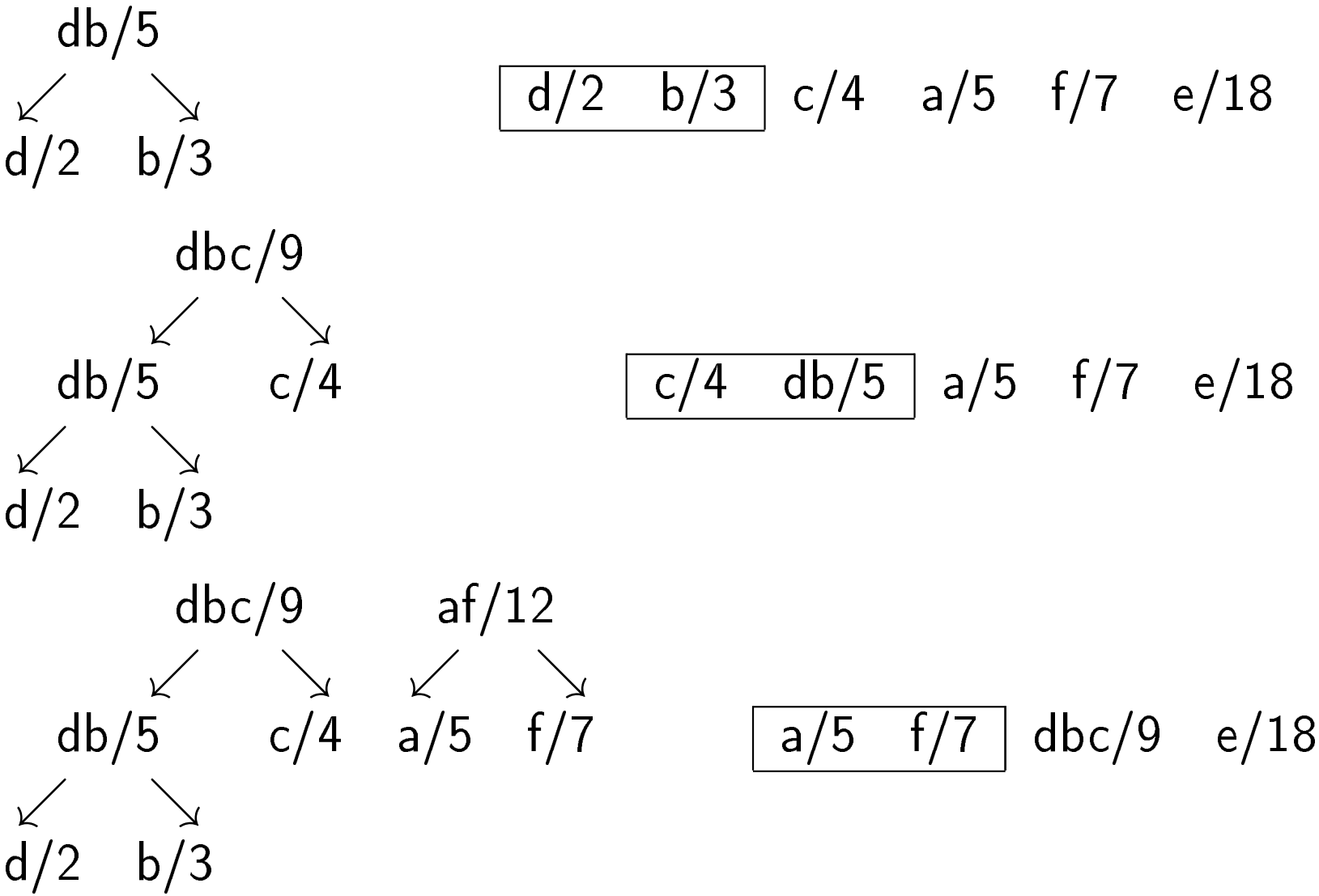
Cas de candidats au groupement de même poids :

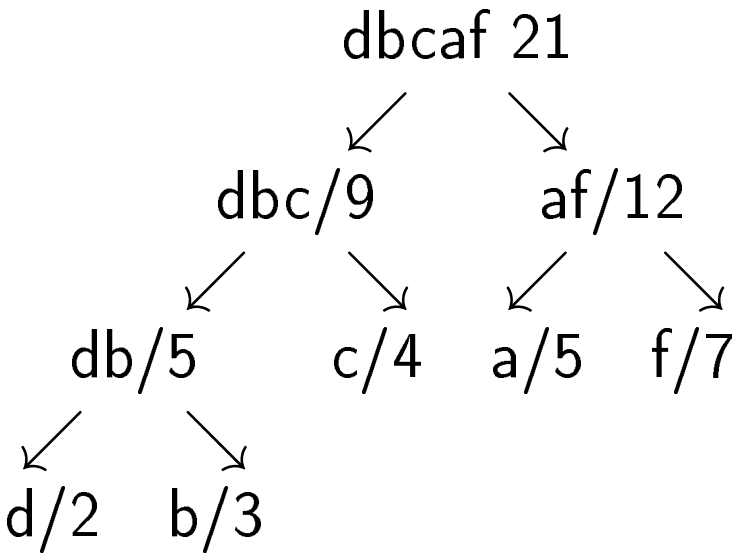
- existence de plusieurs arbres de Huffman
- codes différents et taux de compression identiques

L'arbre de Huffman donne les longueurs de code des symboles à partir desquelles on peut générer les codes et l'arbre canoniques.

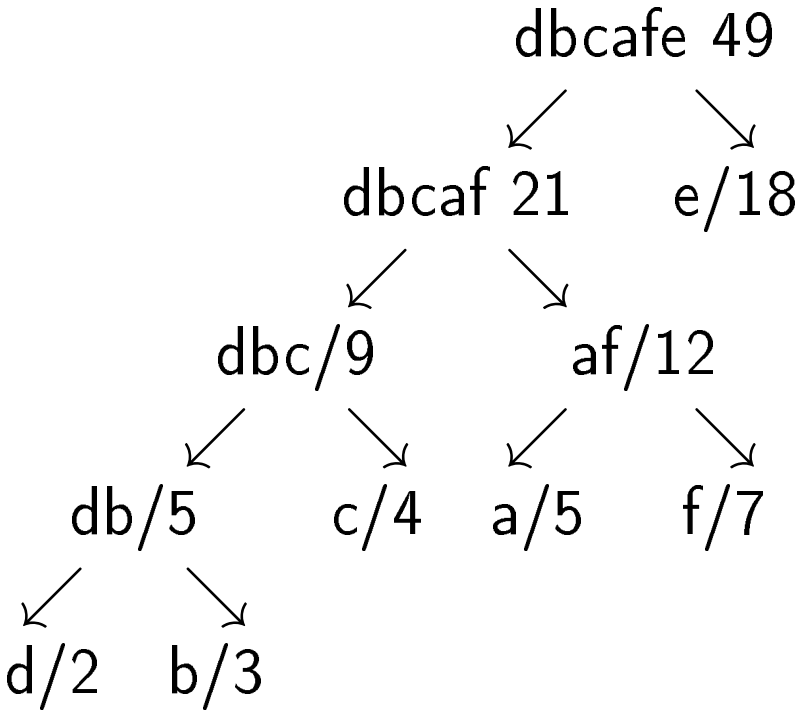
Le codage est optimal, mais $Lmax \in [n-1, N-1]$.

Exemple d'arbre de Huffman



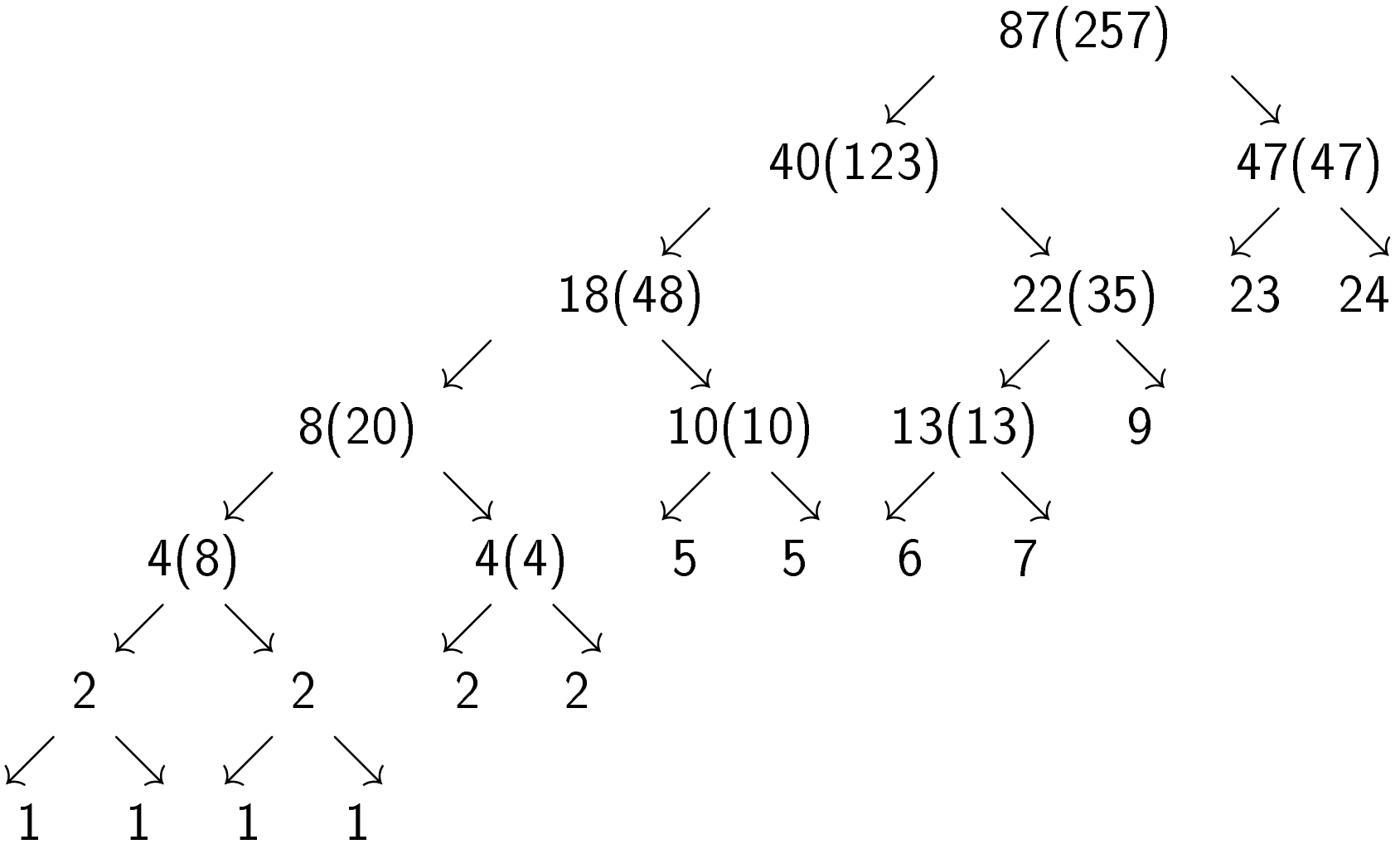


dbc/9	af/12	e/18



e/18	dbcaf/49

Un autre exemple d'arbre (occurences et coûts)



Compression (Huffman non canonique)

1. analyse statique : calcul + tri des nombres d'occurence de symboles
2. construction de l'arbre
3. génération d'une desription de l'arbre ou de ses prémisses (tables des occurences) dans l'en-tête du fichier compressé
4. transcodage + écriture, symbole par symbole, par parcours de l'arbre.

Décompression (Huffman non canonique)

1. lecture de l'en-tête du fichier compressé et reconstruction de l'arbre.
2. lecture bit à bit et décodage des symboles :
   - départ à la racine de l'arbre
   - sur chaque bit lu, transition d'un niveau dans l'arbre
   - arrivée sur feuille : émission symbole + retour à la racine

Prétraitement : Move To Front

Après chaque lecture d'un symbole s le codage des symboles est mis à jour. Les symboles de codes inférieurs au code actuel de s sont décalés d'un cran et s est déplacé en tête (code 0).

Exemple : la chaîne "klkllkabbbaaxzxzxz" comprend 3 portions comprenant chacune une répétition de paires de symboles.

Sans MTF : fréquence égale ($1/6 \simeq 17\%$) des codes des 6 lettres.

Avec MTF : les lettres soulignées sont recodées 0 ou 1 (0 et 1 représentent à eux deux $2/3 \simeq 67\%$ des symboles à compresser).

Exemple MTF : "fcbbeghf"

Alphabet de 8 symboles codés sur 3 bits : a à h									
Codes binaires	Deux derniers symboles lus								
	""	"f"	"fc"	"cb"	"bb"	"be"	"eg"	"gh"	"hf"
000	a	f	c	b	b	e	g	h	f
001	b	A	F	C	c	B	E	G	H
010	c	B	A	F	f	C	B	E	G
011	d	C	B	A	a	F	C	B	E
100	e	D	d	d	d	A	F	C	B
101	f	E	e	e	e	D	A	F	C
110	g	g	g	g	g	g	D	A	a
111	h	h	h	h	h	h	h	D	d

f	c	b	b	e	g	h	f	
101	010	001	001	100	110	111	101	codage initial
101	011	011	000	101	110	111	101	codage mtf

Prétraitement : Run Length Encoding (RLE)

Compression de suites de symboles identiques :

- plages de 0 dans section data (.o , exécutables),
- zones d'image de couleur uniforme.

Ajouter une marque de répétition qui sera suivie (dans les données compressées) d'un nombre de répétitions :
x x x . . . x x (n fois) $\rightarrow$ rep(x,n)

- Code spécial disponible (ex : rep = 0 ou EOT en ASCII) :
  - x x x x x x $\rightarrow$ x *Rep* 5
  - x x $\rightarrow$ x *Rep* 1
- Répétition implicite sur double symbole :
  - $z\ x\ x\ \underline{x\ x\ x\ x}\ y\ \rightarrow\ z\ x\ x\ +4\ y$
  - $z\ x\ x\ y\ \rightarrow\ z\ x\ x\ 0\ y$ (*aucun y proche*)
  - $\underline{\leftarrow\ y\ z\ x\ x\ y}\ \rightarrow\ y\ z\ x\ x\ -3$ (delta codable vers y déjà traité).

Forme canonique de l'arbre de codage

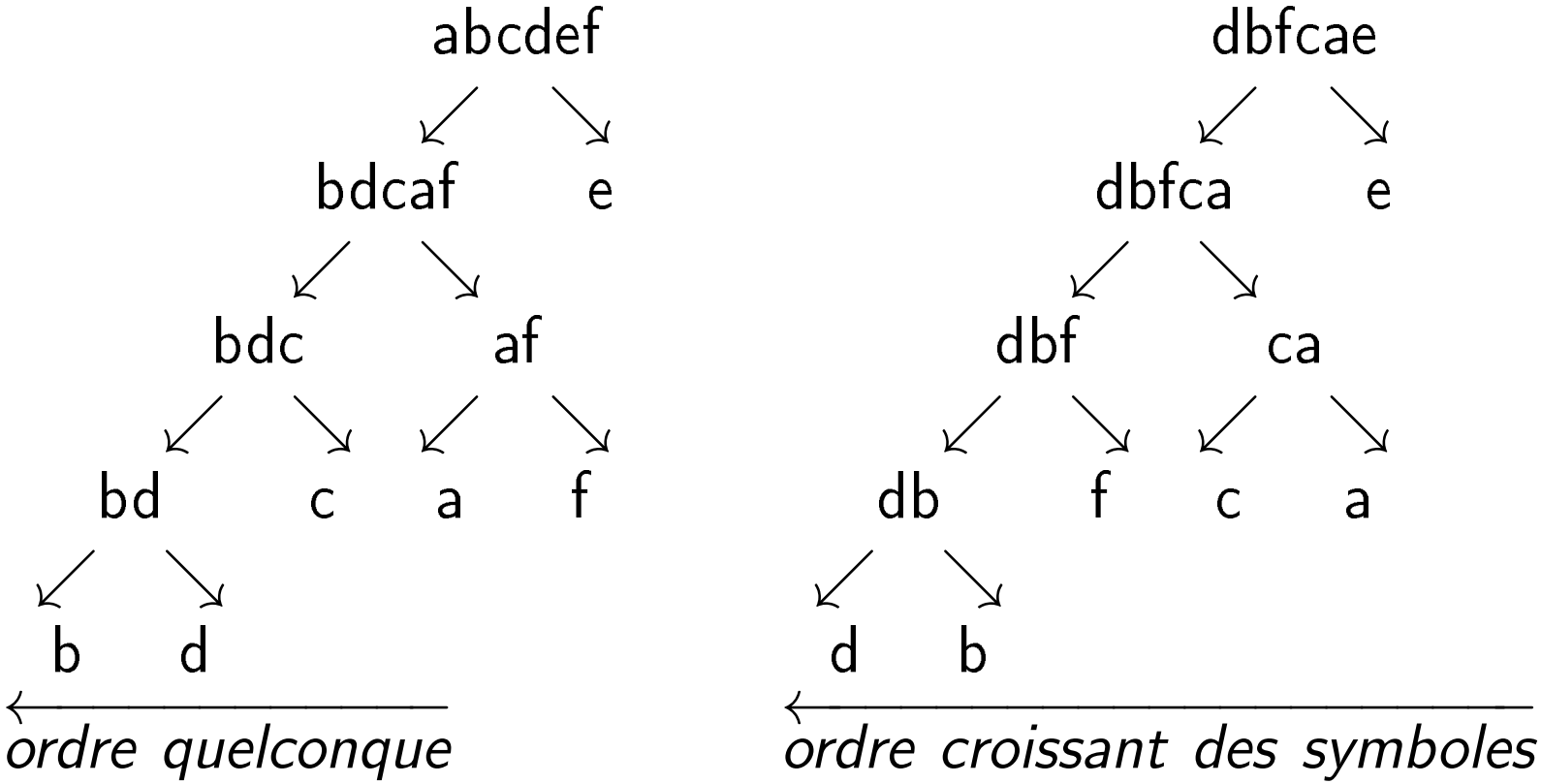
Tous les arbres obtenus par permutation(s) de symboles entre feuilles de même profondeur (même longueur de code) donnent la même taille de données compressées (hors description de l'arbre).

Forme canonique de l'arbre : à profondeur égale

- le code des feuilles précède celui des sous-arbres
- l'ordre des codes de même longueur respecte l'ordre des symboles associés.

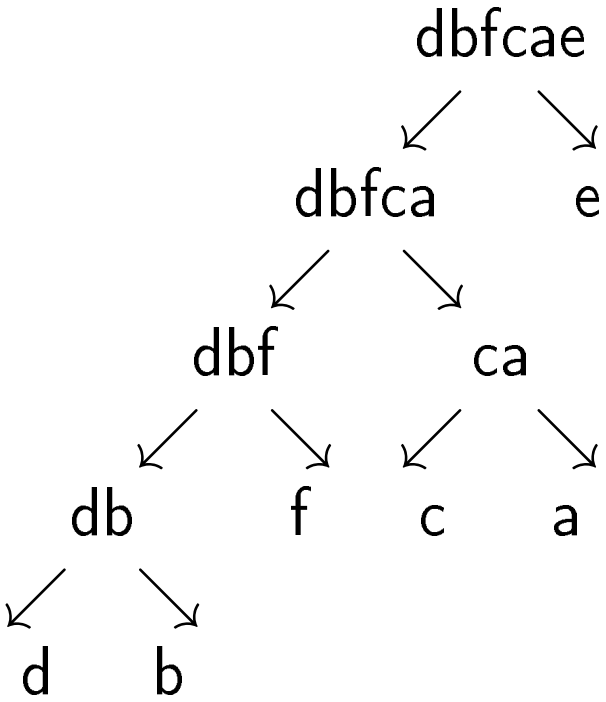
- Il suffit de passer au décodeur une table donnant la longueur du code (pas le code complet ou la fréquence) de chaque symbole.
- Cette table des longueurs de codes peut être compressée en RLE (intéressant pour des suites de symboles de fréquence nulle).

Arbres binaires équivalents



b : 1111, d : 1110, c : 110, a : 101, f : 100, e : 0 (ordre quelconque)
d : 1111, b : 1110, f : 110, c : 101, a : 100, e : 0 (ordre canonique)

Forme canonique de l'exemple



Feuille	Longueur	Code
a	3	100
b	4	1110
c	3	101
d	4	1111
e	1	0
f	3	110

Table triée par symboles : 3,4,3,4,1,3

Table triée d'abord par longueurs : (e,1) (a,3) (c,3) (f,3) (b,4) (d,4)

Les étapes de compression en mode statique

1. prétraitement optionel des données : MTF
2. prétraitement optionel des données : RLE
3. analyse statique : calcul + tri des fréquences de symboles
4. détermination des longueurs des codes des symboles
   - longueurs optimales : construire arbre de HUFFMAN
   - longueurs bornées : algorithme PACKAGE-MERGE
5. génération de arbre/codes canoniques des symboles
6. écriture de table des longueurs (après compression RLE optionnelle)
7. transcodage + écriture, symbole par symbole.

Les étapes de décompression en mode statique

1. lecture (et décompression éventuelle) de la table des longueurs de codes.
2. reconstruction des codes et de l'arbre canoniques
3. lecture bit à bit et décodage des symboles :
   - départ à la racine de l'arbre
   - sur chaque bit lu, transition d'un niveau dans l'arbre
   - arrivée sur feuille : émission symbole + retour à la racine
4. opération inverse de prétraitement optionnel : RLE
5. opération inverse de prétraitement optionnel : MTF

Reconstruire codes et arbre : données initiales

Départ : taille et contenu de la table des longueurs

- N = Cardinal(alphabet) : nombre total de symboles (présents ou non). Exemple typique : N = 257 (256 valeurs d'octet + EOF).
- Longueur[i], $0 \leq i \leq N-1$: longueur du code du symbole i. Longueur=0 pour un symbole inutilisé.

Données simples à extraire des données initiales :

- Lmax : longueur maximale de code (parmi les Longueur[i]).
- Symb[j] : ensemble (trié) des symboles s dont Longueur[s] = j.
- NbSymb[j] : Cardinal(Symb[i]).

Reconstruire codes et arbre : variables

- p : profondeur courante
- c : code courant
- s : symbole courant
- n : nœud courant
- anciens : ensemble de nœuds groupements existants
- nouveaux : nœuds et feuilles en cours de création

Initialisations :

- p=c=0
- anciens=Symb[0]={racine}

Reconstruire codes et arbre : algorithme

Initialisations

Pour p parcourant [1...Lmax] :

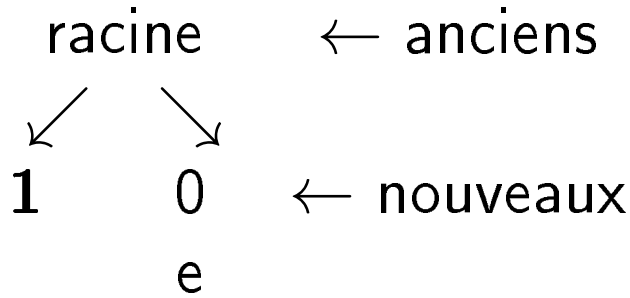
1. Créer nouveaux = ensemble trié de tous les fils de anciens
2. c = c « 1 (ajouter un 0 à droite au code courant)
3. Pour chaque symbole s de Symb[p] :
   - n=retirer_premier_nœud(nouveaux)
   - associer_symbole_nœud_code(s,n,c)
   - c=c+1
4. anciens=nouveaux

Le code associé au premier nœud groupement de ancien est le préfixe du code de la première feuille à l'étape suivante.

A chaque étape $c \leftarrow 2 * (c + NbSymb[p])$.

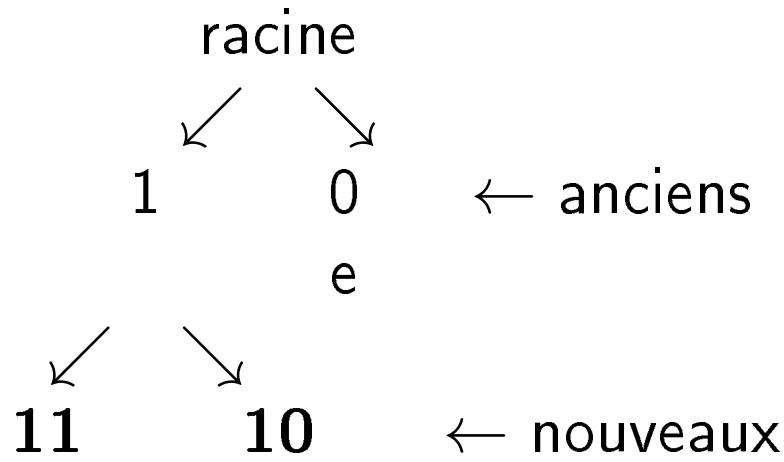
Reconstruire codes et arbre : exemple

p=1, Symb[1]={e}, NbSymb[1]=1, c début = 0



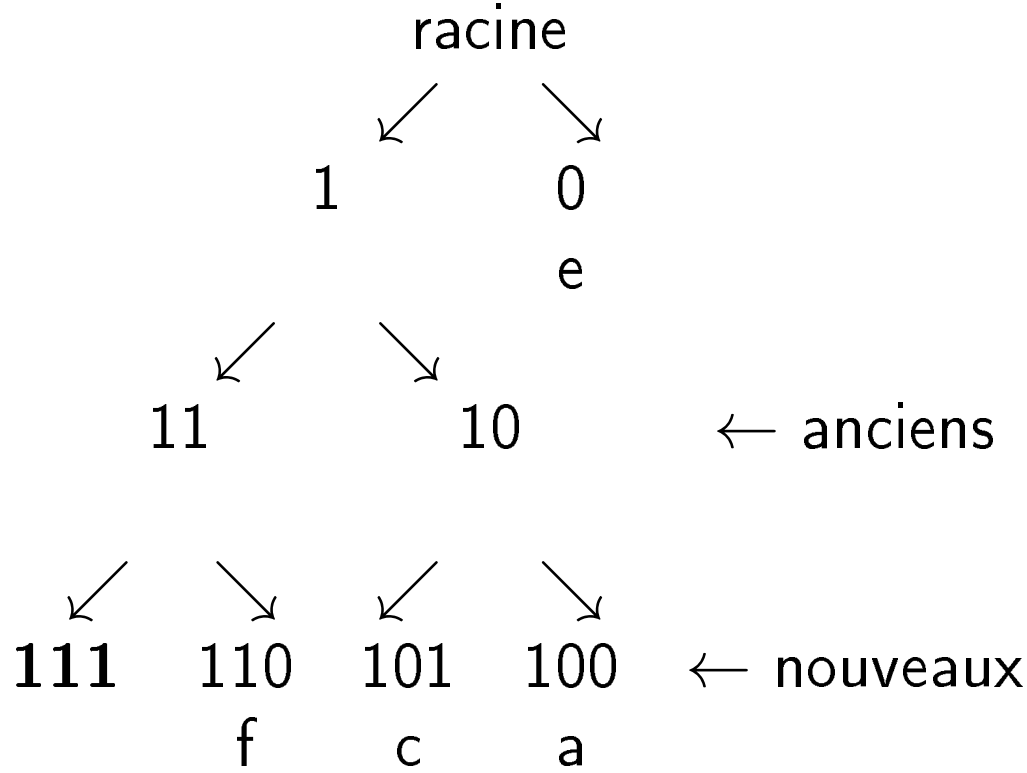
c fin = 2 (10) = $2 * (0 + 1)$

p=2, Symb[2]={}, NbSymb[2]=0, c début = 2



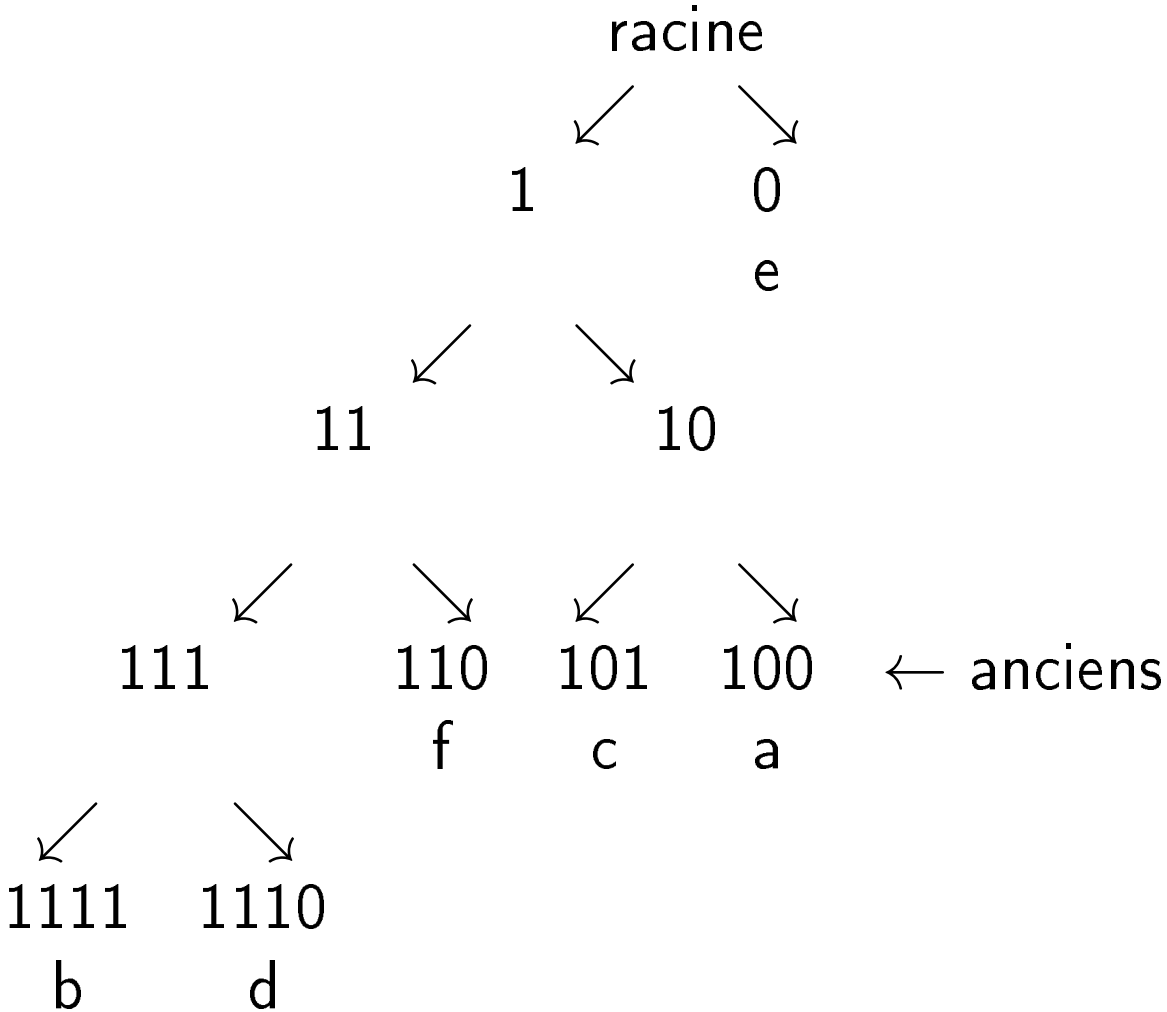
c fin = 4 (100) = $2 * (2 + 0)$

p=3, Symb[3]={a,c,f}, NbSymb[3]=3, c début = 4



c fin = 14 (1110) = $2 * (4 + 3)$

p=4, Symb[4]={b,d}, c début = 14



Description RLE condensée de l'arbre canonique

On suppose qu'on n'ajoute pas de symbole marque de fin de fichier.

Dans le cas d'un fichier binaire, l'en-tête du compressé contient une table des longueurs pour les 256 symboles possibles, soit 256 octets.

Si 'a' à 'f' seuls symboles présents, la table contient :

1. 97 longueurs 0 dans l'intervalle de symboles absents [0,'a'[
2. 6 longueurs des symboles présents ['a','f'] : 3,4,3,4,1,3
3. 153 longueurs 0 dans l'intervalle de symboles absents]'f',255]

La table des longueurs peut être transmise sur 12 octets au format RLE : 0,0,**95**,3,4,3,4,1,3,0,0,**151**

Propriétés de l'arbre à construire

Les symboles de poids nul sont ignorés pour la construction de l'arbre

En entrée : une liste de N paires (symbole, poids non nul).
Liste triée par ordre croissant de poids puis de symbole.

Conventions de vocabulaire et propriétés de l'arbre :

- Feuille : nœud feuille de l'arbre, associé à un symbole.
- Nœud : implicitement nœud autre que feuille ou racine. Chaque nœud est un groupement de sous-arbres.
- Degré 2 : chaque nœud a deux fils
- Composé de N feuilles + $N-2$ nœuds + la racine
- $2^{n-1} \leq N < 2^n \Rightarrow n-1 \leq \textit{hauteur de l'arbre} \leq n$

Principe de construction de l'arbre binaire

Etat de départ : N paires, $2^{n-1} \leq N < 2^n$
arbre de degré N tiré de la liste des paires :
racine + N feuilles, aucun nœud groupement.

La séquence suivante réduit de un le degré de l'arbre.
La répéter pour créer $N - 2$ nœuds.

- Choisir deux fils (gauche, droit) de la racine et les en détacher
- Créer un nouveau nœud pour les regrouper : augmente de un la profondeur de toutes leurs feuilles.
- Attacher le nouveau nœud à la racine.

Etat final : arbre binaire = racine + N feuilles + N-2 nœuds

Arbre de profondeur bornée

On peut décider de privilégier une compression potentiellement suboptimale pour limiter Lmax (par exemple à des entiers courts sur 16 bits) et accélérer le transcodage. C'était important à l'époque des premiers microrordinateurs à microprocesseurs 8/16 bits).

Algorithme de choix de l'arbre optimal de profonseur bornée : Package-Merge.
Question intéreressante : évolution du taux de compression avec la réduction de hauteur de l'arbre ?

En pratique un entier 32 bits sera toujours suffisant pour stocker le code d'un octet. Pour dépasser une hauteur d'arbre de 32, il faut un nombre d'occurences faramineux d'occurences des octets à code court (taille de fichier au-delà des limites de systèmes de fichiers usuels).