Le codage de Huffman

Irena.Rusu@univ-nantes.fr
LINA, bureau 123, 02.51.12.58.16

Sommaire

- Les changements de représentation
- Un problème de codage
- Les codes-préfixes
- L’algorithme de Huffman
- Et le décodage ?

Sommaire

- Les changements de représentation
- Un problème de codage
- Les codes-préfixes
- L’algorithme de Huffman
- Et le décodage ?

Les changements de représentation

- Ce qu'on cherche :
  - « Garder » le contenu (ou l'information) : être capable de le retrouver
  - Changer la forme
- C'est une opération courante :
  - Traduction de documents
  - Changement de devises (sauf commission ...)
  - Numérisation d'une image, etc.
- Ce qu'on vise
  - S'adapter à un contexte (langue, pays …)
  - Vivre avec son temps (tous au numérique …)

Mais aussi …

Simplifier sa vision/compréhension de certaines choses

- Cette possibilité est très utilisée en algorithmique
- Des problèmes sont déclarés faciles ou difficiles parce qu'ils sont équivalents (c.à.d. sont les mêmes à une représentation près).à un problème déjà identifié comme facile/difficile
- Des solutions sont trouvées par réduction à un autre problème : en changeant de représentation on « tombe » sur un problème déjà connu (et résolu aussi bien que possible);
- Exemple :

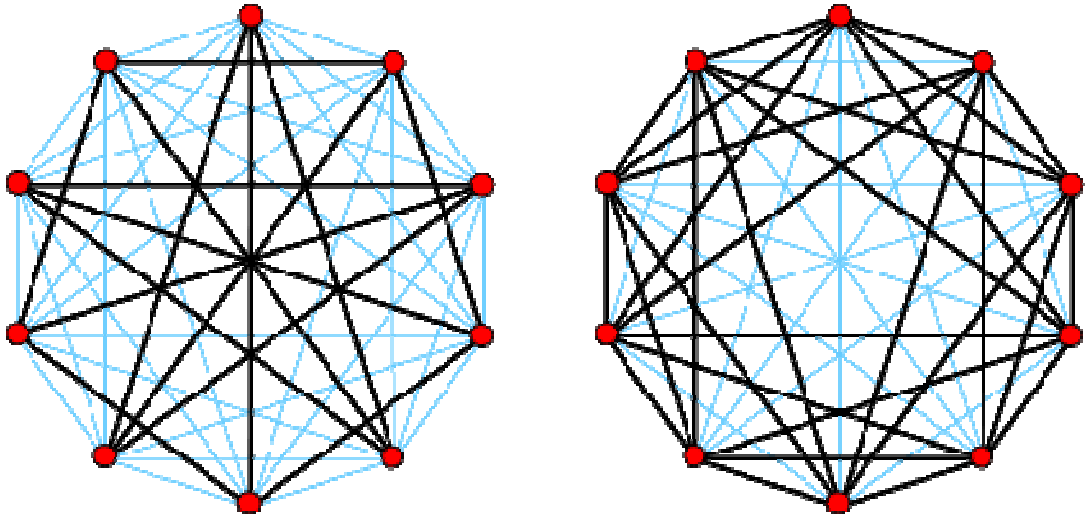
Organisation d'un diner

Entrée : n convives, et leurs in-amitiés 2 à 2

Sortie : un plan de table circulaire tel que les voisins soient amis

Réduction à un problème très connu

- Graphe des in-amitiés vs. graphe des amitiés



- Le problème devient :

  Entrée : un graphe G (le graphe des amitiés, à droite)

  Sortie : un cycle dans G qui contient tous les sommets exactement 1 fois

  → C'est le problème du voyageur de commerce (version circulaire et avec toutes les arêtes du même poids)

Source : Wikipedia

Qu’en déduit-on ?

- Problème très difficile : algorithme efficace (polynomial) peu probable.
- ... Mais beaucoup d’études
  - Algorithmes d’approximation
  - Cas particuliers (plus) faciles
  - Diverses heuristiques assez rapides
  - Solveurs
- Quelques exemples :
  - Parcours de 13 509 villes aux USA: résolu en 1998
  - Parcours de 15 112 villes en Allemagne : résolu en 2001 (110 processeurs à 550 Mhz, Universités de Rice et Princeton, temps total d’utilisation des processeurs : 22.6 années)
- Conclusion pour l’organisation du diner : plus on a d’amis, plus on mange tard.

Et aussi …

Adapter/améliorer/jouer/résoudre … bref, tout (ou presque)

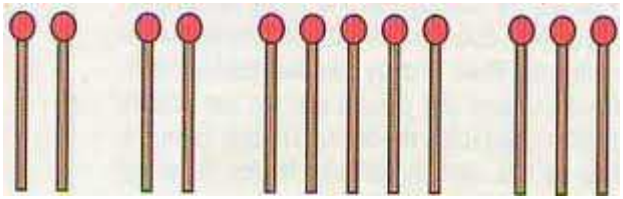
- Exemple : Jeu de Nim
  - N allumettes, 2 joueurs
  - Enlever 1, 2, 3 allumettes tour à tour
  - **But : ne pas être le dernier**

- **A chaque étape, en fonction de son propre niveau de connaissance du jeu, nous avons une autre représentation du jeu**

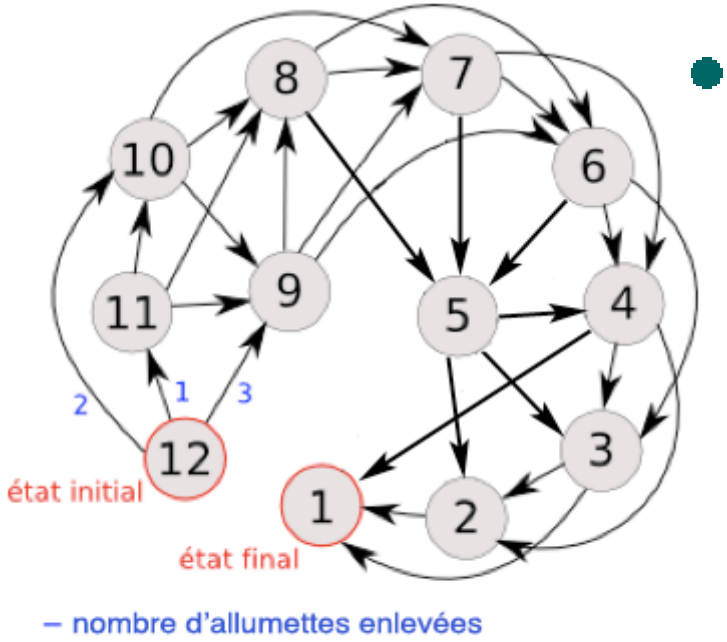
Et aussi …

Adapter/améliorer/jouer/résoudre … bref, tout (ou presque)

- Niveau débutant :



- Niveau intermédiaire



- Niveau expert (quand c'est son tour) :
  - **n≠ 4k+1, je gagne**
  - **n= 4k+1, je perds (probablement)**

Sommaire

- Les changements de représentation
- Un problème de codage
- Les codes-préfixes
- L’algorithme de Huffman
- Et le décodage ?

Un problème de représentation

Codage de Huffman

Entrée : un texte T sur un alphabet donné $\mathcal{A}$

Sortie : un (en)codage de T tel que

- Le code résultant ait une taille minimum (optimalité)
- Le codage et le décodage prennent peu de temps (efficacité)
- (évidemment) Il n'y ait pas de perte d'information → le codage est un changement de représentation de l'information contenue dans T

- Utilisation :
  - Transfert de données
  - Archivage

Principes

- Deux balayages du texte T, de longueur n:
  - 1ère passe : calculer la fréquence de chaque lettre de l'alphabet $\mathcal{A}$
  - 2ème passe : attribuer un code binaire à chaque lettre, en fonction de la fréquence. pour obtenir un encodage S

Note : chaque 0 ou 1 sera représenté sur 1 bit.

- **Remarques**
  - Si code de la même longueur k pour chaque lettre :

$$|S| = kn \text{ bits}$$

  - Pour exploiter la fréquence, les lettres plus fréquentes devraient avoir des codes binaires plus courts

Exemple

- Fichier de 100 000 caractères
- Fréquences observées des 6 lettres du texte :

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5
Mot de code de longueur fixe	000	001	010	011	100	101
Mot de code de longueur variable	0	101	100	111	1101	1100

- Avec code de longueur 3 : |S| = 300000 bits
- Avec code de longueur variable :

|S|= 45000x1+(13000+12000+16000)x3+(9000+5000)x4

= 224000 bits

Sommaire

- Les changements de représentation
- Un problème de codage
- Les codes-préfixes
- L’algorithme de Huffman
- Et le décodage ?

Les codes préfixes

- Code (ou codage) préfixe

  Codage où aucun code binaire utilisé n'est inclus dans un autre code binaire utilisé.

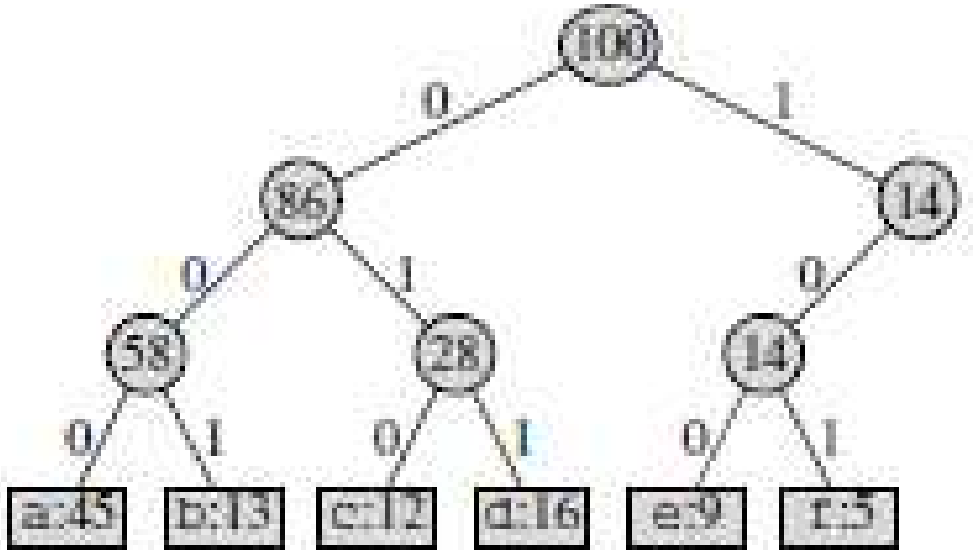
Théorème. La compression maximale possible pour un texte donné peut toujours être obtenue à l'aide d'un codage préfixe.

- Avantages des codes préfixes :
  - Optimalité
  - Efficacité du codage et décodage (pas d'ambiguïté).
- Exemple : S=10111010100 se décode facilement

Arbres binaires (1)

- C’est une représentation (encore une !) d’un codage préfixe
- Les arêtes portent les bits, les feuilles contiennent les caractères encodés (un par feuille)

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5
Mot de code de longueur fixe	000	001	010	011	100	101



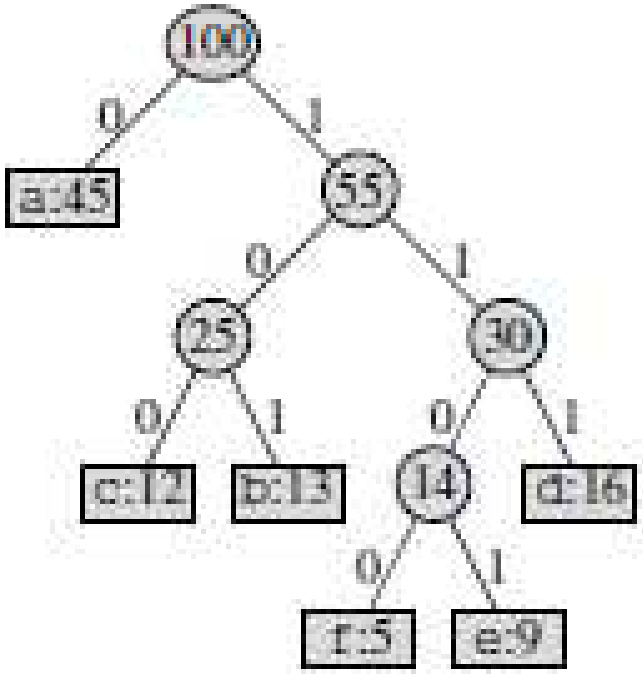
$$|S| = \Sigma_{x \text{ lettre}} p_A(x) f(x)$$

où $p_A(x)$ est la profondeur de x dans l’arbre A

Arbres binaires (2)

- C’est une représentation (encore une !) d’un codage préfixe
- Les arêtes portent les bits, les feuilles contiennent les caractères

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5
Mot de code de longueur variable	0	101	100	111	1101	1100



$|S| = \Sigma_{x \text{ lettre}} p_A(x) f(x)$

où $p_A(x)$ est la profondeur de x dans l’arbre A

Arbres binaires et codage optimal

- Arbre binaire complet : tout nœud interne a 2 enfants.

Lemme. L'arbre binaire représentant un codage préfixe optimal est un arbre binaire complet.

- Codage précédent de longueur fixe : pas optimal.
- Codage précédent de longueur variable : à voir.

Lemme. L'arbre binaire représentant un codage préfixe optimal a $|\mathscr{A}|$ feuilles et $|\mathscr{A}|$-1 nœuds internes.

Sommaire

- Les changements de représentation
- Un problème de codage
- Les codes-préfixes
- L’algorithme de Huffman
- Et le décodage ?

Algorithme de Huffman – Historique

- Auteur: David Albert Huffman (1925-1999)
- Ecrit lors de sa thèse de doctorat au MIT.
- Publié en 1952:

  D.A. Huffman, *A method for the construction of minimum-redundancy codes*, Proceedings of the Institute of Radio Engineers, 1952, pp 1098-1102.

Photo par Don Harris

- Le codage obtenu est optimal pour les hypothèses indiquées (fréquences connues, codage caractère par caractère).
- Ratio de compression: de 20% à 90% en fonction des textes.
- .. Mais le ratio de compression n'est pas optimum. Des méthodes plus évoluées (p. ex. LZ77) basées sur une analyse approfondie du texte et un codage contextuel font mieux.

Algorithme de Huffman – principes (1)

- $|\mathcal{A}|$ feuilles initialement toutes « visibles » (≅ pas encore résolues)
- $|\mathcal{A}|$-1 « fusions » de deux nœuds « visibles »
- Une « fusion » rend invisibles (≅ résolus) les deux nœuds fusionnés et crée un nouveau nœud qui, lui, est visible.

- Pré-condition pour la fusion no. i (et post-condition pour fusion i-1) :

  Le nombre de nœuds visibles est $|\mathcal{A}|$-(i-1).

- Donc, post-condition pour la fusion no. $|\mathcal{A}|$-1 :

  Le nombre de nœuds visibles est 1 (la racine).

Algorithme de Huffman – principes (2)

- Le résultat est un arbre binaire, dont les lettres sont dans les $|\mathcal{A}|$ feuilles → arbre de Huffman

- Toute arête vers un fils est étiquetée 0 ou 1
- L'étiquetage peut être fait au hasard, mais en général on choisit 0 (gauche) et 1 (droite).

- **Choix glouton** : les deux nœuds visibles à fusionner à chaque étape sont ceux de fréquences les plus basses.

Algorithme de Huffman

- Entrée : fréquences $f(a_i)$ des lettres $a_1, \ldots, a_n$ présentes dans T.
- Sortie : arbre binaire correspondant à un codage préfixe pour T

$V \leftarrow \Phi$

pour i de 1 à n **faire**

Créer un arbre à 1 nœud v_i contenant la valeur $f(a_i)$

$freq(v_i) \leftarrow f(a_i)$; $V \leftarrow V \cup \{v_i\}$

finpour

tant que $|V| \geq 2$ **faire**

Soient $G,D \in V$, $G \neq D$, t.q. $freq(G), freq(D) \leq freq(v), \forall v \in V-\{G,D\}$

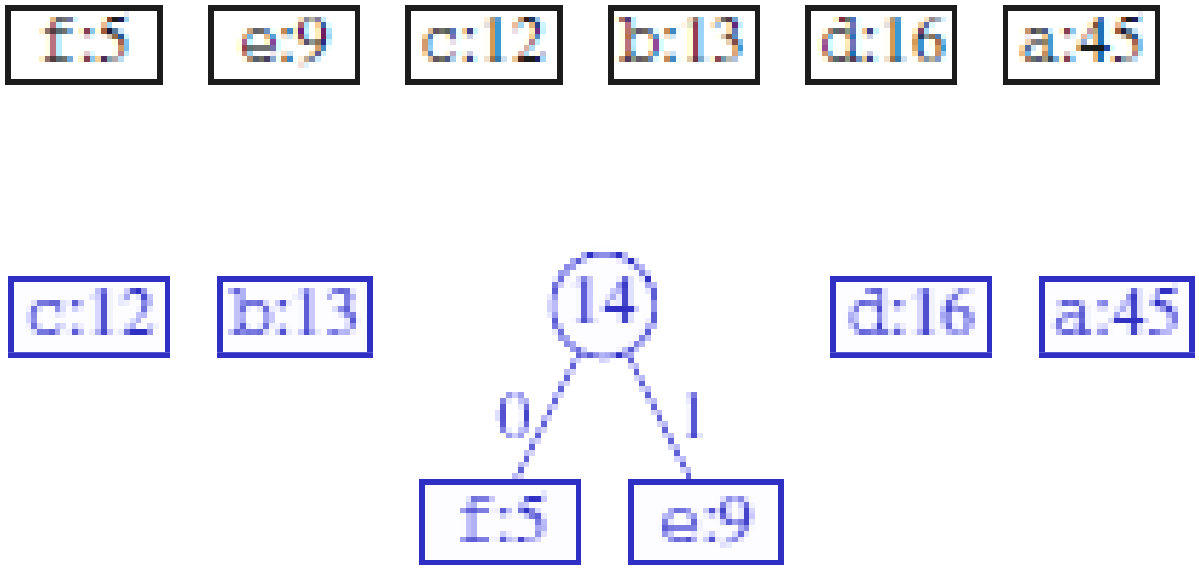
Créer un nouveau noeud R et soient G, D ses fils

étiquette(RG)←0; étiquette(RD)←1

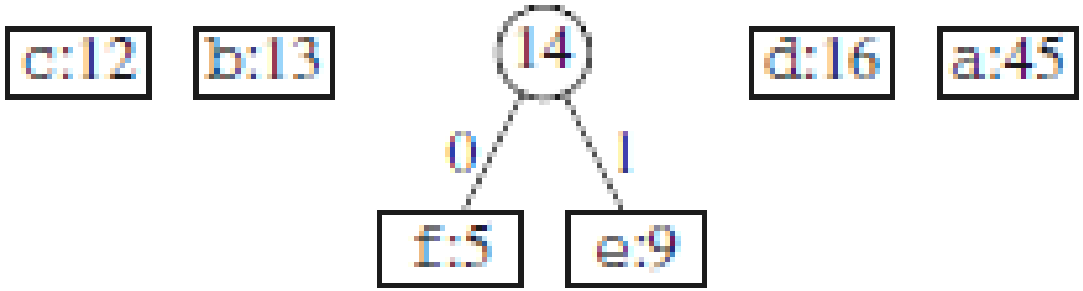
$freq(R) \leftarrow freq(G) + freq(D)$; $V \leftarrow V - \{G,D\} \cup \{R\}$

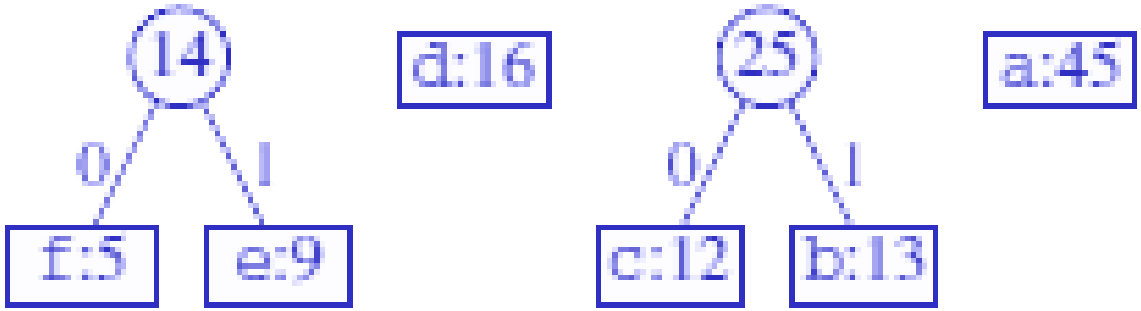
fintantque

Un exemple (1)

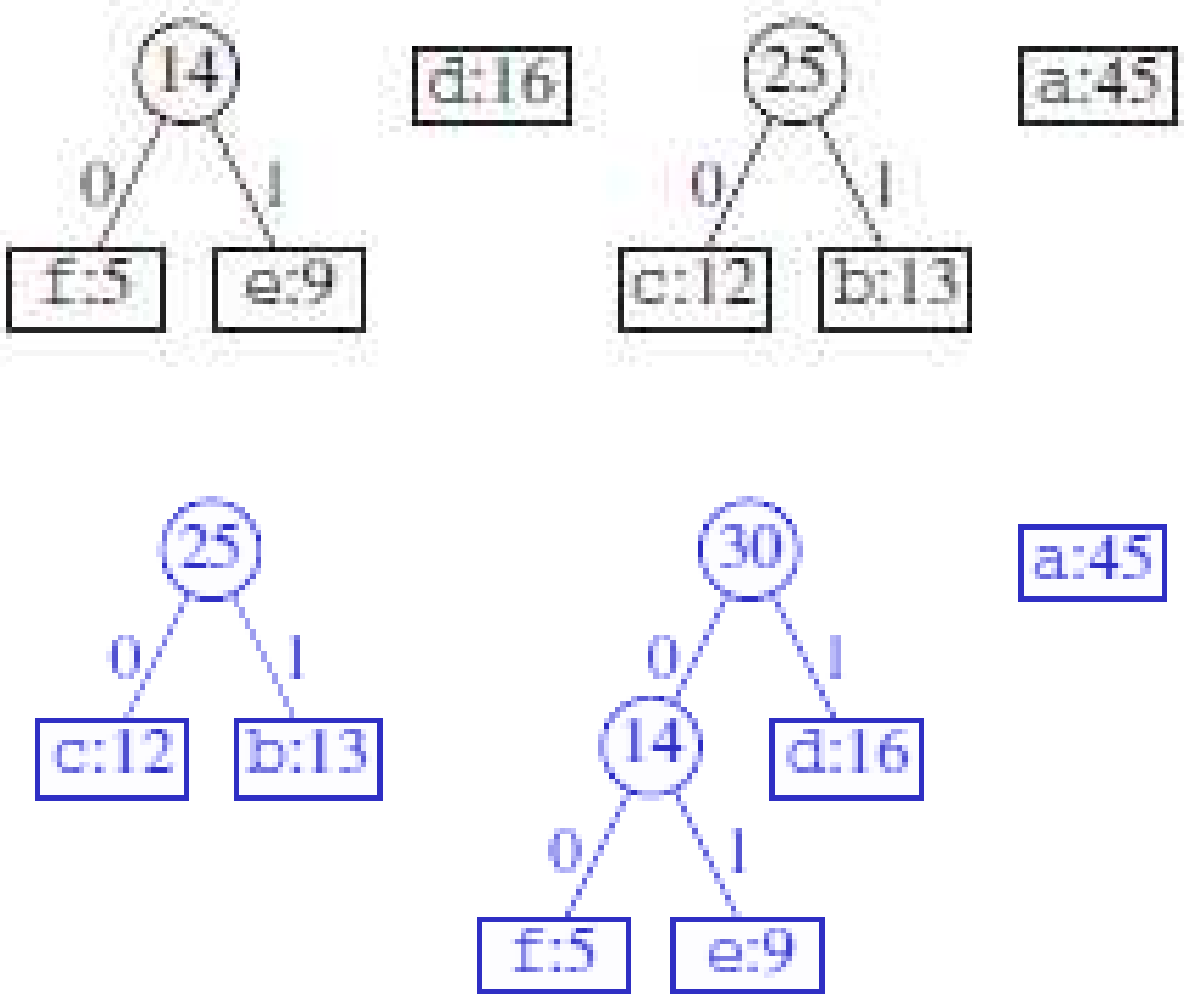


Un exemple (2)

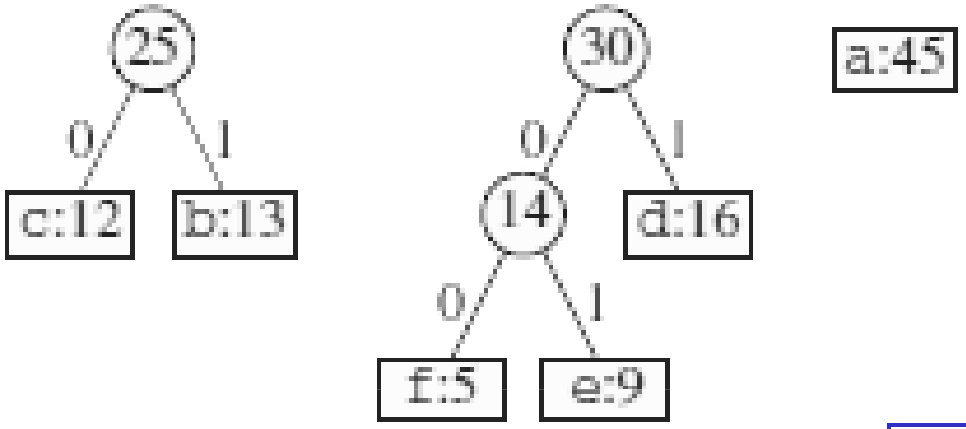


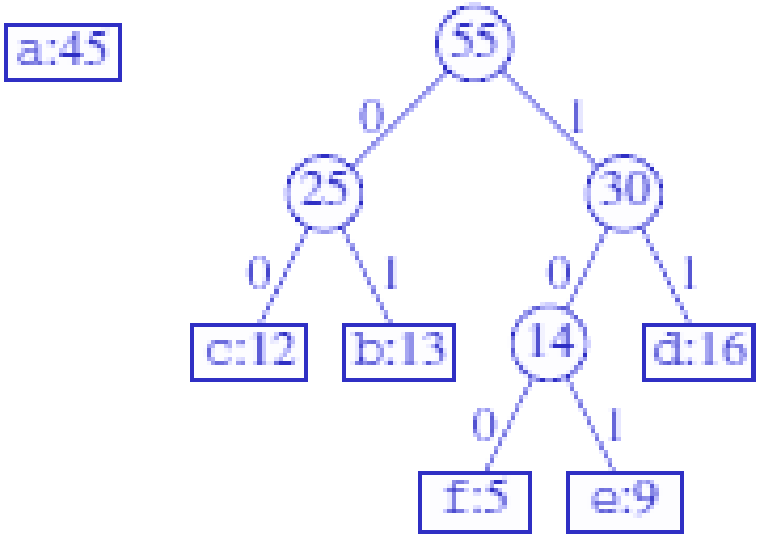


Un exemple (3)

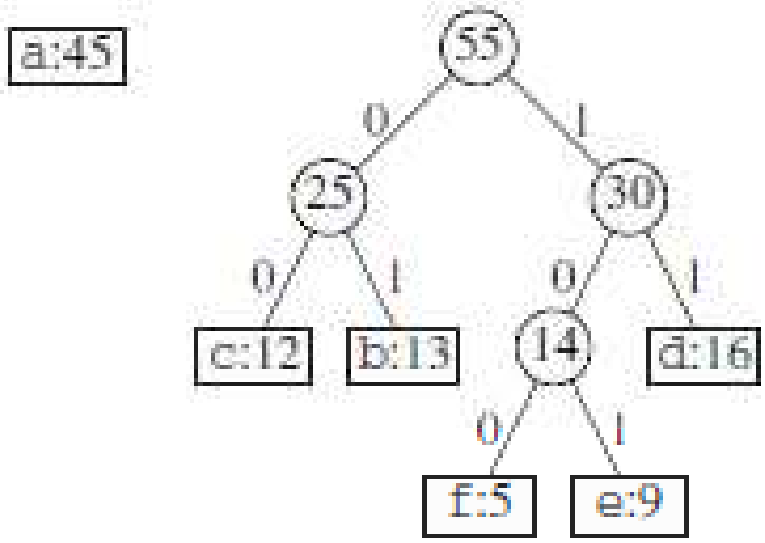

14
0
1
f:5
e:9
d:16
25
0
1
c:12
b:13
a:45
25
0
1
c:12
b:13
30
0
1
14
d:16
0
1
f:5
e:9
a:45

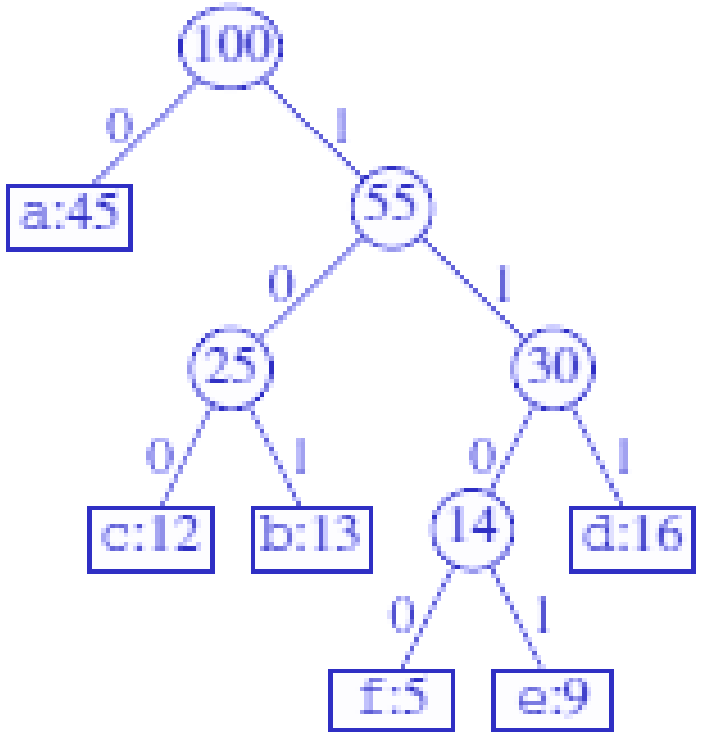
Un exemple (4)


25
0
1
c:12
b:13
30
0
1
14
d:16
0
1
f:5
e:9
a:45


a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9

Un exemple (5)


a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9


100
0
1
a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9

Terminaison, correction et optimalité

- Terminaison
  - le nombre de nœuds de V diminue de 1 à chaque étape
- Correction
  - L'arbre est complet (R a exactement deux fils G, D)
  - Toutes les lettres sont des feuilles, donc le codage est préfixe
- Optimalité
  - on montre qu'il existe toujours (pour tout ensemble de nœuds visibles) un arbre optimal dont les nœuds G,D de l'algorithme sont frères.→ le choix glouton ne réduit pas la généralité de la recherche
  - on montre que la « substitution » de G et D par le nœud R à la fin de la boucle **tant que** n'altère pas l'optimalité
  - donc, un raisonnement inductif de haut vers le bas assure l'optimalité.

Complexité (1)

- Ce n'est pas un algorithme glouton standard !
- Standard :

1. Classer les éléments de E par ordre d'intérêt décroissant

$$e_1, e_2, \ldots, e_n$$

2. $F \leftarrow \Phi$;
3. Pour i de 1 jusqu'à n faire

   si $FU \{e_i\}$ est une solution partielle

   alors

   $F \leftarrow FU\{e_i\}$

   finpour

L'ensemble des éléments, tout comme leur intérêt, n'est pas connu à l'avance.

Le classement en O(nlog n) à l'avance est impossible.

Il faut le refaire à chaque étape.

Complexité (2)

Algorithme de Huffman

	Complexité
$V \leftarrow \Phi$	
pour i de 1 à n **faire**	n exécutions
Créer un arbre à 1 nœud v_i contenant la valeur $f(a_i)$	O(1)
$freq(v_i) \leftarrow f(a_i)$; $V \leftarrow V \cup \{v_i\}$	O(1)
finpour	
tant que $\|V\| \geq 2$ **faire**	n-1 exécutions
Soient $G,D \in V$, $G \neq D$, t.q. $freq(G), freq(D) \leq freq(v)$, $\forall v \in V-\{G,D\}$	????
Créer un nouveau noeud R et soient G, D ses fils	O(1)
étiquette(RG)←0; étiquette(RD)←1	O(1)
$freq(R) \leftarrow freq(G) + freq(D)$; $V \leftarrow V - \{G,D\} \cup \{R\}$	O(1)
fintantque	

Complexité (3)

- Quelle implémentation pour V pour
  - Trouver l'élément de freq() minimum très rapidement
  - Enlever l'élément de freq() minimum très rapidement
  - Insérer un nouvel élément très rapidement ?
- Plusieurs choix :
  - Tas binaire min : arbre, stocké convenablement dans un tableau, où la valeur des éléments augmente lorsqu'on s'éloigne de la racine.
  - AVLs
  - Arbres rouges et noirs

→ O(log n) à l'intérieur de la boucle tant que

→ complexité de l'algorithme de Huffman en O(n log n).

Sommaire

- Les changements de représentation
- Un problème de codage
- Les codes-préfixes
- L’algorithme de Huffman
- Et le décodage ?

Décodage (1)

- Une solution
  - Transmettre le texte encodé S et l'arbre binaire A représentant le codage
  - Pour décoder, lire depuis le début courant du texte et en parallèle dans l'arbre depuis la racine jusqu'à rencontrer une feuille

- Problèmes
  - Comment représenter l'arbre binaire de manière compacte ?
  - L'arbre étant souvent déséquilibré, la recherche du caractère correspondant à un code demande de descendre beaucoup dans l'arbre → lent !

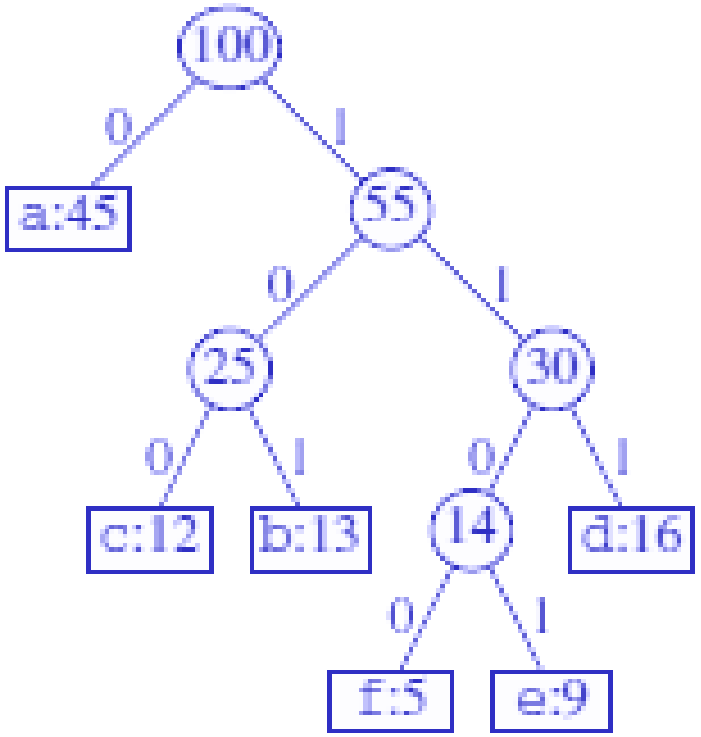
Décodage (2)

- Une autre solution
  - Transmettre le texte encodé S et les triplets (caract., longueur, code),
  - Soit :
    - Reconstruire l'arbre à partir des triplets
    - Décoder à l'aide de l'arbre, comme avant (lent …)
  - Soit :
    - Parcourir le texte, et en parallèle
    - Chercher dans la table/liste/AVL … des triplets
- Problèmes
  - Comment représenter les triplets de manière compacte ?
  - La recherche a un coût

Une réponse possible – Huffman canonique

- Arbre de Huffman canonique : arbre de Huffman dans lequel la profondeur des feuilles ne baisse jamais dans un parcours préfixe de l'arbre.

Nœud	Profondeur
a:45	1
c:12	3
b:13	3
f:5	4
e:9	4
d:16	3



- Ce n'est pas un arbre canonique

Construction d'un arbre de Huffman canonique

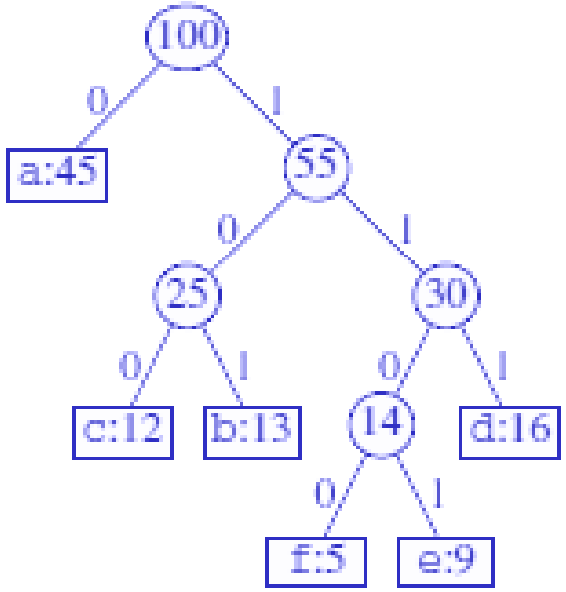
- Construire l'arbre de Huffman pour avoir les profondeurs des feuilles :

  Prof : prof(a_1), prof(a_2), … prof(a_n)

- Ordonner Prof par ordre croissant → ProfOrd
- Construire (algorithme glouton) l'arbre **C** dont les profondeurs des feuilles dans un parcours préfixe sont dans l'ordre ProfOrd
- Etiqueter les arêtes de **C** selon la convention: 0 vers le sous-arbre gauche, 1 vers le sous-arbre droit
- Placer chaque a_i sur une feuille de **C** selon l'ordre décroissant de leurs fréquences

Un exemple

Arbre de Huffman non-canonique



Arbre de Huffman canonique

ProfOrd: 1, 3, 3, 3, 4, 4

Propriétés

- Le premier code est une suite de 0
- Les codes de la même longueur sont consécutifs
- Passage du dernier code de niveau n vers le premier code de niveau n+1 :

Code(n+1,1) = 2(Code(n,dernier)+1)

Equiv : ajouter 0 au code de taille n suivant le dernier utilisé

De plus : **Code(n+2,1) = 2^2(Code(n,dernier)+1) etc.**

→ Pas besoin de vraiment reconstruire l'arbre

Avantages lors du décodage

- Peu d'information à transmettre :
  - Min: longueur du code le plus cours
  - Max : longueur du code le plus long
  - Distribution : nombre de feuilles à chaque niveau

  Pour notre exemple :
  - Min: 1
  - Max : 4
  - Distribution : 1, 0, 3, 2

- Simplifie le décodage (non montré ici)

Conclusion

- L'algorithme de Huffman est un exemple classique d'algorithme glouton
- … mais pas standard dans le sens défini, puisque l'ordonnancement par ordre décroissant de préférence ne peut pas être fait à l'avance
- La construction de l'arbre canonique se fait aussi de manière gloutonne
- … L'ordonnancement est donné à l'avance, mais il n'y a pas de demande d'optimisation (juste de construction)

- Les algorithmes gloutons ont donc quasiment autant de variantes que de problèmes à résoudre.