« Codage Compression Cryptographie »

1

UPMC – M1 – CCC

Cours de Compression

Michèle Soria

Bibliographie

M. Nelson *La Compression de données : texte, images, sons*, Dunod

D. Salomon *Data Compression : The Complete Reference*, Springer

Compression Statistique

2

1. Algorithme de Huffmann statique
   - Principes et Définitions
   - Construction de l'arbre de Huffman
   - Optimalité du code
   - Compression : algorithme et complexité
   - Déompression : algorithme et complexité
2. Huffmann Adaptatif
   - Principes et Définitions
   - Modification de l'arbre de Huffman adaptatif
   - Compresseur et décompresseur

Compression statistique – Huffman

3

Symbole fréquent == code court,
Symbole rare == code long

Au mieux : coder un symbole avec le nombre de bits d'information qu'il contient $log_2(1/p(i))$, (mais ici nombre de bits est *entier*).

Connaître les fréquences
- Table de probabilité universelle (d'ordre 0, 1, ...)
- Table de probabilité statique pour chaque texte à compresser : meilleure compression mais surcoût (calcul des fréquences + transmission de la table) $\longrightarrow$ Algorithme de Huffmann
- Fréquences calculées au fur et à mesure (codage d'un symbole évolue) $\longrightarrow$ Algorithme de Huffmann adaptatif
  Adaptativité inversible (fonction de décompression)

Code préfixe

4

Huffman : code préfixe (décodage) de longueur variable (statistiques)

Alphabet $A = \{a_1, \ldots, a_n\}$. Codage $C : A \rightarrow \{0,1\}^+$.
P ensemble de mots de $\{0,1\}^+$, codages des caractères de A.

Définition : *P code préfixe* $\stackrel{Def}{=}$ aucun mot de P n'est préfixe propre d'un mot de P : $\forall w_1 \in P,\ w_1 w_2 \in P \rightarrow w = \varepsilon$.

Propriété : *Tout code préfixe est uniquement décodable*

Aucun code d'un caractère n'est préfixe propre d'un code d'un autre caractère : $C(xy) = C(x)C(y)$.

ARBRE DIGITAL

5

Définition : *arbre digital (trie)* $\stackrel{Def}{=}$ arbre binaire t.q. chaque feuille contient le codage de son chemin à partir de la racine (0 pour lien gauche, 1 pour droit).

Propriété : *P est un code préfixe ssi il est constitué des codes des feuilles d'un arbre digital.*

Définition : Arbre digital *complet* $\stackrel{Def}{=}$ Code préfixe *complet*

Remarques : Pour tout $a \in A, C(a)$ est un mot sur $\{0,1\}^+$, de longueur $L(a)$. $L(a)$ est égal à la profondeur de la feuille codant a dans l'arbre digital.

CODE MINIMAL

6

Texte $T \in A^*$. Codage $C : T \to T(C) = T_C$.

Définitions

- *Fréquence* $f(a_i)$ $f(a_i) = \#a_i$ dans T.
- *Taille* de $T(C)$: $||T_C|| = \sum f(a_i)|C(a_i)|$, $a_i \in A$
- *C code minimal pour T* ssi $||T_C|| = min\{||T_\gamma||, \gamma : A \to \{0,1\}^+\}$

Propriété : *C minimal pour T* $\Longrightarrow$
$f(a_{i_1}) \leq f(a_{i_2}) \leq \ldots \leq f(a_{i_n})$ *et* $|C(a_{i_1})| \geq |C(a_{i_2})| \geq \ldots \geq |C(a_{i_n})|$

CONSTRUCTION DE L'ARBRE DE HUFFMAN

7

Exemple : $T = abracadabra$,
$f(a) = 5, f(b) = 2, f(r) = 2, f(c) = 1, f(d) = 1$

Construction d'un arbre digital

- Peigne par fréquences croissantes,
  ex. 1, 1, 2, 2, 5 $||T_C|| = 23$
- Utiliser les branchements
  ex. 1, 1, 2, 2, 3 $||T_C|| = 20$ (au lieu de 21 pour peigne)

Principe de construction d'un arbre de Huffman

- Unir 2 sous-arbres de "poids minimal"
- Deux degrés de liberté : choix des 2 sous-arbres + Gauche/Droite

Exemple : plusieurs arbres de Huffman pour le texte $T = abracadabra$

EXEMPLE

8

ARBRE DE HUFFMAN : ALGORITHME

ENTRÉE : Fréquences $(f_1, \ldots, f_n)$ des lettres (a_i) d'un texte T.
SORTIE : Arbre digital donnant un code préfixe minimal pour T.

1. Créer, pour chaque lettre a_i, un arbre (réduit à une feuille) qui porte comme poids la fréquence $f(i)$
2. Itérer le processus suivant
   - choisir 2 arbres G et D de poids minimum
   - créer un nouvel arbre R, ayant pour sous-arbre gauche (resp. droit) G (resp. D) et lui affecter comme poids la somme des poids de G et D,
3. Arrêter losqu'il ne reste plus qu'un seul arbre : c'est l'arbre de Huffman

STRUCTURATION DES DONNÉES

Création de l'arbre de Huffman contrôlée par une *file de priorité* (TAS) – éléments = arbres, et clés = poids attachés aux arbres (fréquences des lettres pour les feuilles et poids cumulés pour les arbres construits)–.

```
;Alphabet*entier*entier*...* entier→ ArbreHuffman
Fonction Huffman(A,f1,...,fn)   ;fi ≠ 0 fréquence de ai dans T
FP=construireTas((f1,a1),...,(fn,an))
pour i de 1 à n-1
    ;extraireMinTas rend l'arbre min, et le tas réorganisé
    (G,FP1)=extraireMinTas(FP)
    (D,FP2) =extraireMinTas(FP1)
     R=ABValué(G,D,f(G)+f(D))
     FP=ajouterTas(R,FP2)
finPour
retourne extraireMin(FP)
```


AUTRE VERSION

Définition récursive de la construction de l'arbre

```
; ; Huffman : Alphabet*entier*entier*...* entier→ ArbreHuffman
Fonction Huffman(A,f1,...,fn)
    ;construireTas : Alphabet*entier*...* entier→ TAS[ArbreHuffman]
retourne  ArbreH(construireTas((f1,a1),...,(fn,an))

; ; ArbreH : TAS[ArbreHuffman]→ ArbreHuffman
Fonction ArbreH(FP)
Si FP contient 1 seul arbre retourne cet arbre
Sinon
   Soient   (G, FP1)=extraireMinTas(FP)
            (D, FP2) =extraireMinTas(FP1)
retourne ArbreH(ajouterTas(ABValué(G,D,f(G)+f(D)),FP2))
```


COMPLEXITÉ DE LA CONSTRUCTION

1. Construire le tas initial : $(O(n))$ (ou $O(n \log n)$ en ligne)
2. A chaque itération
   - extraire 2 fois l'arbre de poids min et réorganiser le tas : $O(\log n)$
   - fabriquer un nouvel arbre à partir des 2 précédents : $O(1)$
   - rajouter ce nouvel arbre au tas : $O(\log n)$
3. $n-1$ itérations

La construction de l'arbre de Huffman est donc en $O(n \log n)$ comparaisons.

OPTIMALITÉ DU CODE

13

Taille du texte compressé :

$||T_h|| = W(H) = \sum_1^n f(i)\text{prof}_h(i) = \sum_1^n f(i)p(i)$

Propriété : *Le code préfixe complet de l'arbre de Huffman est minimal i.e. aucun arbre digital ne compresse mieux le texte T que Huffman(A,(fi))*

ALGORITHME DE COMPRESSION

14

ENTRÉE : Texte T
SORTIE : Texte compressé TC et codage de l'arbre de Huffman

1. Parcourir T pour compter le nombre d'occurrences de chaque caractère
2. Construire l'arbre de Huffman à partir des fréquences des caractères
3. Calculer la table des codes à partir de l'arbre de Huffman
4. Compresser T en concaténant les codes de chaque caractère → TC
5. Coder l'arbre de Huffman de façon compacte

ANALYSE

15

Texte T : N caractères sur alphabet de n lettres,
Codage initial d'un caractère sur k bits

1. Parcours de T : $O(N)$. Stockage des fréquences : $O(n)$,
2. Construction de l'arbre H en $O(n\log n)$,
3. Construction table des codes : $2n-1 = O(n)$. Stockage : $kn + \sum L_i$ bits,
4. Parcours de T : $O(N)$. Taille de TC : $\sum f_i L_i$,
5. Codage de H : Temps $O(n)$. Taille de l'arbre codé $2n-1+kn$ bits.

CODAGE DE DYCK

16

Table des codes $\Longleftrightarrow$ Tous les chemins de l'arbre de Huffman

Mieux : coder un arbre binaire complet (n feuilles et $n-1$ nœuds internes) un mot de Dyck (Lukasiewicz) de longueur $2n-1$.

Parcours préfixe
– nœud interne $\longrightarrow$ 0
– feuille $\longrightarrow$ 1
– De plus chaque 1 est suivi de k bits (code initial du caractère)

( ) - ( ) - (( ) -) -) -

Exemple : 01a01b001d1c1r

Décodage : reconstruire l'arbre : 0 = nœuds internes et 1= feuille, et après chaque 1 lire k bits = codage initial d'un caractère

ALGORITHME DE DÉCOMPRESSION

17

ENTRÉE : Texte compressé TC et codage de l'arbre de Huffman
SORTIE : Texte décompressé T

1. Reconstruire l'arbre de Huffman ($O(n)$)
2. Parcourir TC en suivant l'arbre de Huffman ($O(||TC||)$)
   - suite de bits : de la racine à une feuille de l'arbre
   - produit un caractère de T
   - et l'on poursuit le parcours de TC en recommençant à la racine de l'arbre

STATIQUE ⟶ DYNAMIQUE

18

Incovénients de la compression de Huffman statique
- double parcours du texte (fréquences + compression)
- mémorisation du codage (arbre de Huffman)

Version dynamique – adaptative
- l'arbre de Huffman évolue au fur et à mesure de la lecture du texte et du traitement (compression ou décompression) des caractères.
- l'algorithme de compression (*compresseur*) et l'algorithme de décompression (*décompresseur*) modifient l'arbre de la même façon
- à un instant donné du traitement les 2 algorithmes utilisent les mêmes codes – mais ces codes changent au cours du temps.

PRINCIPE HUFFMAN DYNAMIQUE

19

Compresseur Connaît les codes fixes (k bits) des caractères,
- Initialement, toutes fréquences nulles, et arbreH = caractère spécial #
- A la lecture de x dans T,
  - si prem. occ., retourne *code# dans H + code fixe*, et ajoute x à H
  - sinon retourne *code (variable) de x dans H*

  augmente de 1 la fréquence de x et modifie éventuellement l'arbre H

Décompresseur Connaît les codes fixes (k bits) des caractères
- Initialement lit k bits, et ajoute caractère à arbre "vide" (H =#)
- Puis décode les bits du texte compressé sur H ⟶ feuille
  - si feuille=#, lit les k bits suivants de TC et ajoute le x à H
  - sinon retourne le caractère x correspondant à la feuille

  augmente de 1 la fréquence de x et modifie éventuellement l'arbre de Huffman de la même manière que le compresseur.

ARBRE DE HUFFMAN ADAPTATIF

20

Définition : Numérotation hiérarchique gauche-droite bas-haut (GDBH)
Définition : Un *arbre de Huffman adaptatif* est un arbre de Huffman tel que le parcours hiérarchique GDBH $((x_1,\ldots,x_{2n-1})$ donne la suite des poids en ordre croissant $W(x_1) \leq \ldots \leq W(x_{2n-1})$

Propriété P : *Soit H un arbre de Huffman adaptatif et φ une feuille, de numéro hiérarchique x_{i_0}, dont le chemin à la racine est $\Gamma_\varphi = x_{i_0}, x_{i_1}, \ldots, x_{i_k}$ ($i_k = 2n-1$). Le chemin Γ_φ vérifie* ***P*** *ssi $W(x_{i_j}) < W(x_{i_j+1})$, pour $0 \leq j \leq k-1$.* **N.B.** le sommet x_{i_j+1} suit x_{i_j} dans le parcours GDBH.

Proposition : *soit H un arbre de Huffman adaptatif, φ une feuille de H et Γ_φ son chemin à la racine. Si Γ_φ vérifie la* ***propriété P****, alors l'arbre résultant de l'incrémentation de φ est encore un arbre de Huffman adaptatif.*

EXEMPLES

21

PRINCIPE DE MODIFICATION

22

Modification après lecture d'un caractère de T correspondant à la feuille φ, de chemin Γ_φ à la racine de l'arbre de Huffman

- si Γ_φ vérifie **P**, incrémenter les poids sur ce chemin, sans modifier l'arbre,
- sinon transformer l'arbre pour qu'il vérifie **P**, puis incrémenter les poids sur le (nouveau) chemin de φ à la racine.

Transformation de l'arbre

- soit m le premier sommet de Γ_φ qui ne vérifie pas **P**, et soit f tel que $W(x_m) = W(x_{m+1}), \ldots = W(x_f)$ et $W(x_f) < W(x_{f+1})$ (f est en *fin de bloc* de m : on dispose d'une fonction `finBloc(H,m)` qui renvoie le nœud `f`
- échanger les sous-arbres $A1$ et $A2$ de racines numérotées x_m et x_l
  N.B. On n'échange jamais un nœud avec un de ses ancêtres
- recommencer le même traitement sur le nouveau chemin de la racine de $A1$ à la racine de l'arbre de Huffman

ALGORITHME DE MODIFICATION

23

Feuille spéciale # de fréquence 0, et dont la position varie au cours du temps

```
;Modification : ArbreHuffmanAdaptatif * Caractère → ArbreHuffmanAdaptatif
Fonction Modification (H,s)
Si s n'est pas dans H
   Soit Q le père de la feuille spéciale # dans H
   Remplacer # par un noeud interne de poids 1, dont
      - le fils gauche est la feuille "#"
      - le fils droit est une feuille "s", de poids 1
 Sinon
   Soit Q la feuille correspondant à s dans H
   Si Q est frère de # et pere(Q)=finBloc(H,Q), alors Q=pere(Q) Fi
FinSi
retourne Traitement(H,Q)
```


TRAITEMENT

24

```
;Traitement : ArbreHuffmanAdaptatif * noeud → ArbreHuffmanAdaptatif
Fonction Traitement (H,Q)
Soient Q,Q_i1,...,Q_ik = ChQ le chemin de Q à la racine
et soit x_i0,x_i1,...,x_ik la numérotation GDBH de ChQ.
Si  Q vérifie la propriété P
    ajouter 1 à tous les poids du chemin ChQ
    retourne H
Sinon
   Soit m le premier indice de ChQ t.q. W(x_m)=W(x_{m+1})
   et soit f l'indice de fin de bloc associé à m
   ajouter 1 à tous les poids du chemin de Q à Q_{m}
   Échanger dans H les sous-arbres enracinés en Q_m et Q_f
   retourne Traitement(H,pere(Q_m))
```

Complexité $O(n)$ (hauteur maximale de H)

ALGORITHME DE COMPRESSION

25

ENTRÉE : Texte T
SORTIE : Texte compressé TC

1. H=feuille spéciale #
2. soit s le symbole suivant de T
   - Si $s \in H$, alors transmettre le code de s dans H
     Sinon transmettre le code de # dans H puis le code initial de s
   - Modifier H : `H=Modification (H, s)`
3. Recommencer l'étape 2 tant que T n'est pas vide

Complexité $O(nN)$

Implantation en TME

ALGORITHME DE DÉCOMPRESSION

26

ENTRÉE : Texte compressé TC (et connaît codes fixes des caractères)
SORTIE : Texte décompressé T

1. H= #, décoder k bits $\rightarrow s$, et `H=Modification(H,s)`
2. Lire les bits de TC en suivant H à partir de sa racine $\rightarrow$ feuille
   - Si c'est la feuille # alors lire les k bits suivant $\rightarrow s$
     Sinon $\rightarrow$ caractère s de la feuille
   - `H=Modification(H,s)`
3. Recommencer l'étape 2 tant que TC n'est pas vide

Complexité $O(nN)$

Implantation en TME

CONCLUSIONS

27

- Huffman Dynamique ne calcule pas les fréquences
- Huffman Dynamique "à la volée" : décompression peut commencer avant que compression terminée
- Codage Huffman Statique minimal pour $T_{(f_i)}$, mais doit transmettre l'arbre $\Rightarrow$ augmente taille de l'encodage
- Résultats expérimentaux sur données homogènes : Huffman Dynamique légèrement meilleur que Huffman Statique pour petits fichiers, et bien meilleur pour gros fichiers. (A tester en TME !)