Les algorithmes de compression de données

Site Internet : **www.gecif.net**	Type de document : **Cours**	Intercalaire :	Date :

I – La compression RLE

I – 1 – Principe :

Le codage par plages ou codage par longueur de plage (appelé en anglais **R**un **L**ength **E**ncoding : **RLE**) est un algorithme de compression de données sans perte qui repose sur l'idée de comprimer ***des plages de valeurs identiques*** en signalant ***le nombre de fois qu'une valeur donnée est répétée***.

La méthode de compression RLE (Run Length Encoding) est utilisée par de nombreux formats d'images (BMP, PCX, TIFF). Elle est basée ***sur la répétition d'éléments consécutifs***.

Le principe de base consiste à coder un premier octet donnant le nombre de répétitions (appelée ***occurrence***) d'une valeur puis de le compléter par la valeur à répéter. Ainsi selon ce principe la chaîne "AAAAAHHHHHHHHHHHHHH" compressée par RLE donne "5A14H", soit sur 4 octets : 5 + A + 14 + H (l'occurrence « 14 » étant codée sur un seul octet). Le gain de compression est ainsi de (19-4)/19 soit environ 78.9%. En contrepartie pour la chaîne "REELLEMENT", dans laquelle la redondance des caractères est faible, le résultat de la compression RLE donne "1R2E2L1E1M1E1N1T" ; la compression s'avère ici très coûteuse, avec un gain négatif valant (10-16)/10 soit -60% ! La compression RLE sera réellement efficace seulement pour des données où les octets se répètent un grand nombre de fois, comme par exemple dans le codage des images.

Après compression par l'algorithme RLE on obtient un ensemble d'octets alternant parfaitement un nombre d'occurrences, un caractère à répéter, un nombre d'occurrences, un caractère à répéter, etc. Les occurrences comme les caractères sont chacun codé sur 1 octet (8 bits). La valeur maximale des octets étant 255, si un caractère est répété plus de 255 fois, il faudra le coder sur 4 octets dans la version compressée par RLE. Exemple : imaginons une chaîne de caractères composée de 500 caractères Z consécutifs (elle occupe donc 500 octets si elle n'est pas compressée). Après compression RLE notre chaîne se code 255Z245Z (ce qui veut dire « répéter 255 fois le Z puis répéter 245 le Z » ce qui fait bien 500 caractères Z consécutifs puisque 255 + 245 = 500), soit sur 4 octets seulement. Le gain est alors de (500-4)/500 = 99.2% !

Remarques : les occurrences (ici les nombres 255 et 245) sont codées sur 1 octet (même s'ils sont écrits sur 3 chiffres sur le papier). Pour représenter sur le papier une chaîne de 500 caractères Z on pourra écrire Z…Z avec une accolade horizontale en dessous indiquant l'occurrence de 500 en notant « **500 fois** » (et sans écrire 500 Z !).

I – 2 – Exemples :

Dans le tableau suivant complétez toutes les informations manquantes pour les 6 exemples donnés :

Chaîne non compressée	Codage RLE	Calcul du gain
RRRRRRRNNNNNEEEEEE		
	17K9L3C	
	204B175A247Y	
GGEECCIIFF		
	255D255D78D	
	9Q6T7V8X3M5H	

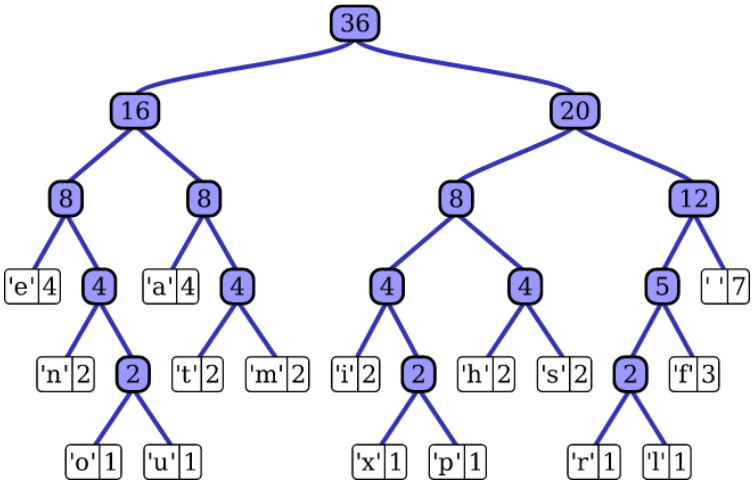
II – Le codage de Huffman

II – 1 – Principe :

Le codage de Huffman est un algorithme de compression de données sans perte. Le codage de Huffman utilise un code à longueur variable pour représenter un symbole de la source (par exemple un caractère dans un fichier). Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents, et un code plus long sera attribué au symboles plus rares.

Cet algorithme de compression de données a été inventé par David Albert Huffman, et publié en 1952.

Le principe du codage de Huffman repose sur la création **d'une structure d'arbre binaire** composée de nœuds pondérés.
Supposons que la phrase à coder est ***« this is an example of a huffman tree »***. On recherche tout d'abord le nombre d'occurrences de chaque caractère. Dans l'exemple précédent, la phrase contient 2 fois le caractère h, 7 espaces, 1 seul caractère x, etc. **Chaque caractère constitue une des feuilles** de l'arbre à laquelle on associe un poids égal à **son nombre d'occurrences**.
L'arbre est créé de la manière suivante : on associe chaque fois les deux nœuds de plus faibles poids, pour donner un nouveau **nœud père dont le poids équivaut à la somme des poids de ses fils**. On réitère ce processus jusqu'à n'en avoir plus qu'un seul nœud : **la racine**. On associe ensuite par exemple le code 0 à chaque embranchement partant vers la gauche et le code 1 vers la droite.



Un exemple d'arbre binaire de Huffman, généré avec la phrase « this is an example of a huffman tree »

Pour obtenir le code binaire de chaque caractère, on remonte l'arbre **à partir de la racine jusqu'aux feuilles** en rajoutant à chaque fois au code un 0 ou un 1 selon la branche suivie. La phrase ***« this is an example of a huffman tree »*** se code alors sur 135 bits au lieu de 288 bits (si le codage initial des caractères tient sur 8 bits par caractère). Il est nécessaire de partir de la racine pour obtenir les codes binaires car sinon lors de la décompression, partir des feuilles peut entraîner une confusion lors du décodage. Pour la décompression il faudra transmettre l'arbre.

II – 2 – Exemples :

Soit le mot « ABRACADABRA ». Ce mot contient …… caractères : il occupe donc …… octets s'il est codé en ASCII.

Étape 1 : donner l'occurrence de chaque lettre :

A : …… B : …… R : …… C : …… D : ……

Étape 2 : construction de l'arbre binaire ci-contre.

On commence par les lettres les moins utilisées qui donnent les premières feuilles, puis on remonte jusqu'à la racine qui contient alors le nombre total de lettres.

Arbre binaire de Huffman pour le mot « ABRACADABRA »

Étape 3 : en consultant l'arbre on en déduit le codage binaire de chaque caractère :

A : …… B : …… R : …… C : …… D : ……

Étape 4 : on en déduit le codage binaire complet du mot « ABRACADABRA » en utilisant la méthode de Huffman :

……

Conclusion : codé en ASCII, le mot « ABRACADABRA » occupe …… bits, et compressé par le codage de Huffman le mot « ABRACADABRA » occupe …… bits. Le gain est alors de ……

Même questions pour le mot « **anticonstitutionnellement** » (ce mot contient …… lettres) :

Étape 1 :

Étape 3 :

Étape 4 :

Conclusion :

Étape 2 : *arbre binaire de Huffman pour le mot « anticonstitutionnellement »*