Institut Villebon Georges Charpak – UE 3.1 Transmission de l'Information

Devoir à la Maison: Compression – Algorithme d'Huffman

On dit souvent qu'au XXIe siècle nous sommes entrés dans l'*Ère de l'Information*. La quantité d'information crée et échangée sous forme numérique est gigantesque: par exemple, en moyenne 5700 tweets sont envoyés chaque seconde (Twitter 2013). Or stocker et envoyer l'information est coûteux en énergie et en matières premières. C'est pourquoi, l'information est souvent *compressée* avant d'être envoyée.

Qu'entends t'on par compression ?

Comme vous le savez l'information peut-être codée sous plusieurs formes. Les ordinateurs représentent les nombres et les lettres en binaire.

Une des représentations les plus simples, la représentation ASCII, code l'alphabet de la manière suivante:

Lettre	Code	Lettre	Code
A	01000001	N	01001110
B	01000010	O	01001111
C	01000011	P	01010000
D	01000100	Q	01010001
E	01000101	R	01010010
F	01000110	S	01010011
G	01000111	T	01010100
H	01001000	U	01010101
I	01001001	V	01010110
J	01001010	W	01010111
K	01001011	X	01011000
L	01001100	Y	01011001
M	01001101	Z	01011010

En ASCII, le message "ALGO" est codé "01000001010011000100011101001111".

Chaque lettre est représentée par 8 bits d'information. En effet le code ASCII

permet de représenter en tout 256 (2^8) caractères: il y a aussi les chiffres, les lettres minuscules, les signes accentués et la ponctuation.

Néanmoins, si notre message ne contient que des lettres en majuscules il est facile de le compresser. En effet, vous remarquerez que les trois premiers bits de chaque code ci-dessus sont identiques. Ces bits sont donc redondants et peuvent être éliminés.

Le facteur de compression est le rapport moyen entre la taille d'un message avant et après compression.

(1) Quel est le facteur de compression obtenu en supprimant les trois premiers bits du code ci-dessus ?

Peut-on encore compresser d'avantage les messages et en particulier les messages écrits en français ? La réponse est OUI, mais avant de comprendre comment, il faut s'intéresser à l'analyse fréquentielle d'un texte...

Analyse des fréquences dans un texte

L'analyse fréquentielle est très utile pour la compression, ainsi que pour la cryptographie. C'est elle qui a permis de casser les premiers codes secrets (pour les curieux, lire l'excellent ouvrage *Histoire des Codes Secrets* de Simon Singh).

L'analyse fréquentielle d'un texte consiste à compter le nombre d'occurrences de chaque caractère dans un texte. (Cela devrait vous rappeler le comptage d'effectifs de poulets).

Ainsi, l'analyse des fréquences de la berceuse bien connue:

```
AU CLAIR DE LA LUNE,
MON AMI PIERROT,
PRETE MOI TA PLUME,
POUR ECRIRE UN MOT
```

Produira la table de fréquence suivante:

espace	E	R	A	I	M	O	U	L	P	T	retour	virgule	N	C	D
12	8	7	5	5	5	5	5	4	4	4	4	3	3	2	1

(2) Écrivez un programme qui prends un texte en entrée et affiche sa table des fréquences à l'écran. Assurez vous que les entrées sont affichées par ordre de fréquence décroissante.

On constate très rapidement que tous les caractères ne sont pas aussi fréquents. En 1952 Huffman a utilisé cette observation pour compresser les textes.

Compression de Huffman

Jusqu'ici nous avons représenté chaque symbole avec le même nombre de bits. Nous avons ici 16 symboles, nous pourrions donc coder "Au clair de la lune" ainsi:

symbole	espace	E	R	A	I	M	O	U
fréquence	12	8	7	5	5	5	5	5
code	0000	0001	0010	0011	0100	0101	0110	0111

symbole	L	P	T	retour	virgule	N	C	D
fréquence	4	4	4	4	3	3	2	1
code	1000	1001	1010	1011	1100	1101	1110	1111

On aurait dans ce cas très exactement besoin de coder 77 symboles, soit $77 * 4 = 312$ bits en tout.

L'idée de Huffman est de coder les symboles très fréquents avec un faible nombre de bits. Essayons par exemple de coder le symbole le plus fréquent "espace" avec 1 seul bit.

symbole	espace	E	R	A	I	M	O	U
fréquence	12	8	7	5	5	5	5	5
code	0	0001	0010	0011	0100	0101	0110	0111

symbole	L	P	T	retour	virgule	N	C	D
fréquence	4	4	4	4	3	3	2	1
code	1000	1001	1010	1011	1100	1101	1110	1111

Est-ce que le code résultant est toujours valide ? Non car maintenant un message peut-être décodé de plusieurs manières (notre code n'est plus une *bijection*). Par exemple le code "01000", peut se lire comme "0-1000 = espace-L" ou "0100-0 = I-espace". Cela est problématique car le récepteur ne peut plus décoder notre message de manière unique...

Comment éviter ce problème, Huffman propose une solution simple: aucun mot du code ne peut commencer comme un autre mot. On appelle cela la règle *préfixe*. Par exemple, pour rendre le code ci-dessus *préfixe* il faut rajouter un 1 en début de tous les codes (sauf celui pour "espace").

symbole	espace	E	R	A	I	M	O	U
fréquence	12	8	7	5	5	5	5	5
code	0	10001	10010	10011	10100	10101	10110	10111

symbole	L	P	T	retour	virgule	N	C	D
fréquence	4	4	4	4	3	3	2	1
code	11000	11001	11010	11011	11100	11101	11110	11111

(3) Montrez que le code préfixe ci-dessus ne peut-être décodé que d'une seule manière.

La propriété préfixe est embêtante: pour raccourcir "espace" on a dû rallonger tous les autres codes ? Est-ce qu'on y gagne ?

Non car: on a gagné 3 bits sur les 12 occurrences de "espace", donc 36 bits au total; mais on a perdu 1 bit sur les 65 autres caractères, donc 65 bits de perdus...

On a été trop brutal. Il faut trouver un meilleur compromis. Si on demandait à Huffman il nous proposerait le code suivant pour notre exemple:

symbole	espace	E	R	A	I	M	O	U
fréquence	12	8	7	5	5	5	5	5
code	000	110	100	0110	0111	0100	0101	1110

symbole	L	P	T	retour	virgule	N	C	D
fréquence	4	4	4	4	3	3	2	1
code	1111	1010	1011	00110	00111	00100	001010	001011

Vous remarquez que les symboles fréquents on des codes "courts" alors que les symboles infréquents on des codes "longs".

(4) En combien de bits peut on représenter "Au clair de la lune" avec le code de Huffman ? Quel est le facteur de compression ? Expliquez pourquoi Huffman raccourcit les codes des symboles fréquents ?

Implantation de l'algorithme de Huffman

Pour trouver un code optimal Huffman propose l'algorithme que nous allons détailler. Huffman a prouvé en 1952 que son algorithme trouve le meilleur facteur de compression pour un message dont la table de fréquences est donnée et le

codage se fait caractère par caractère. Nous ne donnerons pas la preuve ici, car elle fait intervenir des notions mathématiques que vous n'avez pas encore vues. Néanmoins, nous allons voir "avec les mains" l'intuition derrière l'algorithme de Huffman.

L'algorithme de Huffman est constitué de trois grandes étapes:

- *Étape 1*: on construit la table des fréquences du texte à compresser
- *Étape 2*: on construit l'arbre de Huffman
- *Étape 3*: on transforme l'arbre de Huffman en table de code

L'étape 1 a été déjà faite précédemment. Nous allons donc faire les étapes 2 et 3.

Construction de l'Arbre de Huffman

L'arbre de Huffman est binaire: chaque noeud interne a exactement deux fils dans l'arbre. Un noeud de l'arbre sera représenté avec la structure suivante:

```
struct Noeud {
    int frequence;
    char symbole;
    struct * Noeud gauche;
    struct * Noeud droite;
};
```

Nous utiliserons également un tableau nommé `foret` de de type `struct * Noeud` qui pourra contenir au maximum 256 arbres de Huffman.

```
#define MAX_ARBRES 256
struct * Noeud foret[MAX_ARBRES];
```

Les Noeuds dans le tableau `foret` doivent toujours être dans l'ordre des fréquences croissantes.

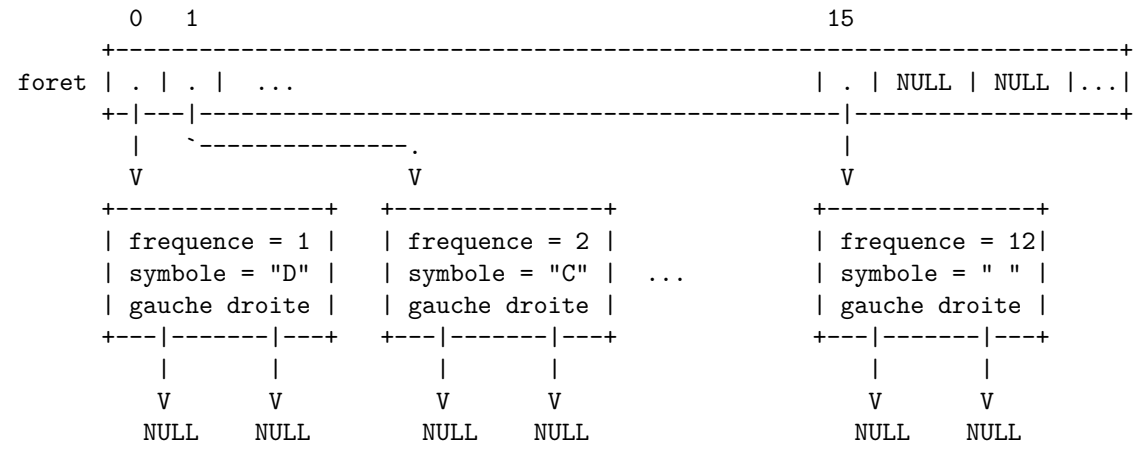
(5) Écrivez la fonction `ajouter` qui permet d'ajouter un Noeud au tableau foret à la bonne position.

(6) Écrivez la fonction `supprimer` qui permet de supprimer un Noeud du tableau foret.

(7) Écrivez une fonction `initialisation` qui prends une table de fréquences en entrée. Pour chaque symbole de la table la fonction fera les actions suivantes:

- Créer un `Noeud` en remplissant les champs symbole et fréquence selon les valeurs du symbole dans la table.
- Ajoute le `Noeud` crée à la table `foret`.

Si vous exécutez la fonction `initialisation` sur “Au clair de la lune”, la mémoire du tableau foret devrait ressembler au schéma suivant:



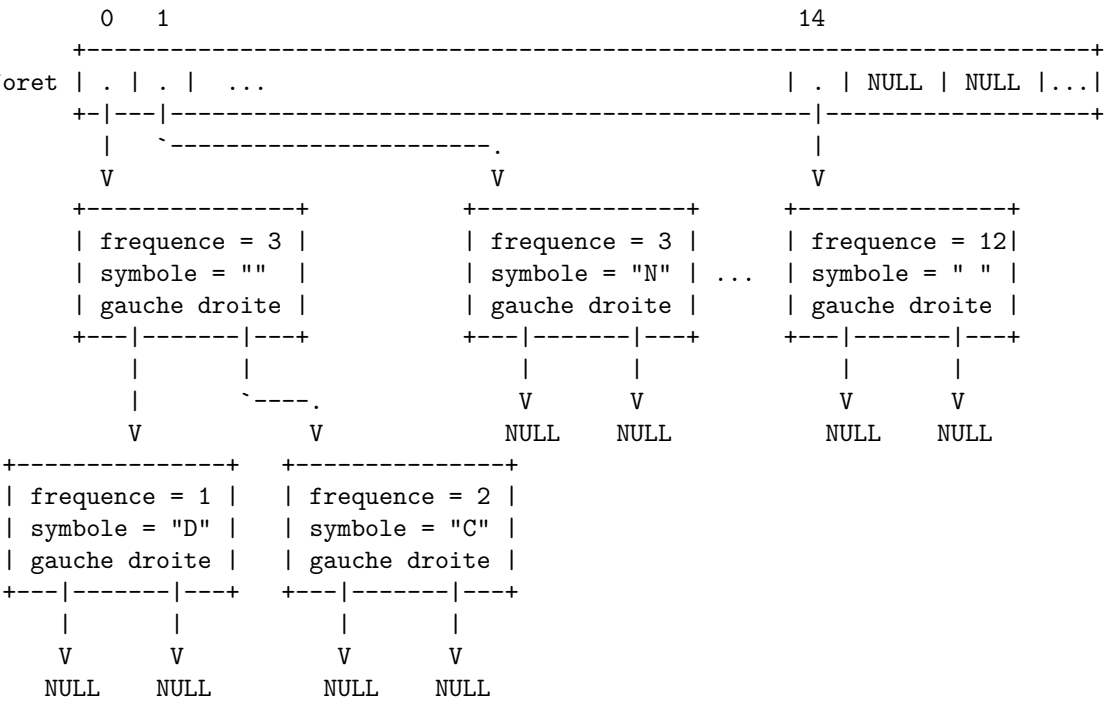
(8) Maintenant écrivez une fonction, `fusion`, qui cherche les deux noeuds dans le tableau foret de plus faible fréquence (les deux premiers) et les fusionne.

Pour fusionner deux noeuds, on fait les étapes suivantes:

- On crée un nouveau `Noeud`, **nouveau**
- La fréquence de **nouveau** est la somme des fréquence des noeuds fusionnés
- Le symbole de **nouveau** est la chaîne vide "".
- Les champs `gauche` et `droite` de **nouveau** pointent vers les noeuds fusionnés.

Après fusion on retire les deux anciens Noeuds du tableau foret et on rajoute au tableau foret le Noeud **nouveau** (à la bonne position).

Par exemple, si on exécute la fonction `fusion` sur l'exemple ci-dessus on obtiendra:



(9) Pour terminer écrivez une fonction `fusion_iteree`, qui répète l'opération fusion jusqu'à ce qu'il ne reste plus qu'un seul noeud dans le tableau `foret`.

Observez que les symboles fréquents se retrouvent proches de la racine de l'arbre ainsi crée.

Création de la table des codes

À la fin de l'étape précédente vous devriez avoir obtenu un arbre proche de celui en figure 1.

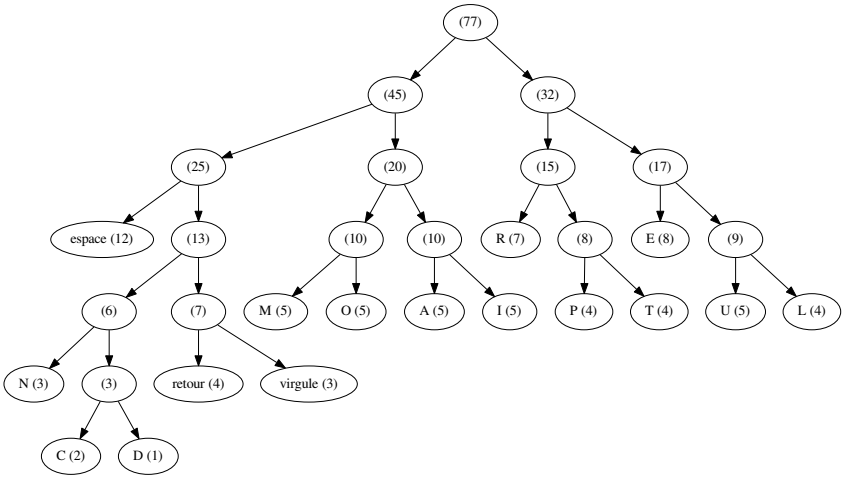


Figure 1: Arbre de Huffman

Pour obtenir le code d'un des symboles il suffit de parcourir le chemin menant jusqu'à lui en partant de la racine. À chaque fois qu'une branche gauche est empruntée, vous ajoutez 0 au code. À chaque fois qu'une branche droite est empruntée vous ajoutez 1 au code.

Par exemple, pour aller en "R" il faut prendre une fois à droite et deux fois à gauche. Le code de "R" est donc 100. Ceci apparaît en figure 2.

(10) Écrivez un programme qui calcule la table des codes à partir de l'arbre de Huffman.

(11) Expliquez pourquoi l'arbre de Huffman génère une table de codage efficace pour la compresssion de texte.

Compression et Décompression

(12) Maintenant que vous savez générer une table des codes idéale, écrivez un programme qui compresse un fichier.

(13) Testez votre compresseur sur plusieurs types de fichiers: texte, image, son, fichier déjà compressé. Quel est le facteur de compression constaté dans chaque cas ? Expliquez.

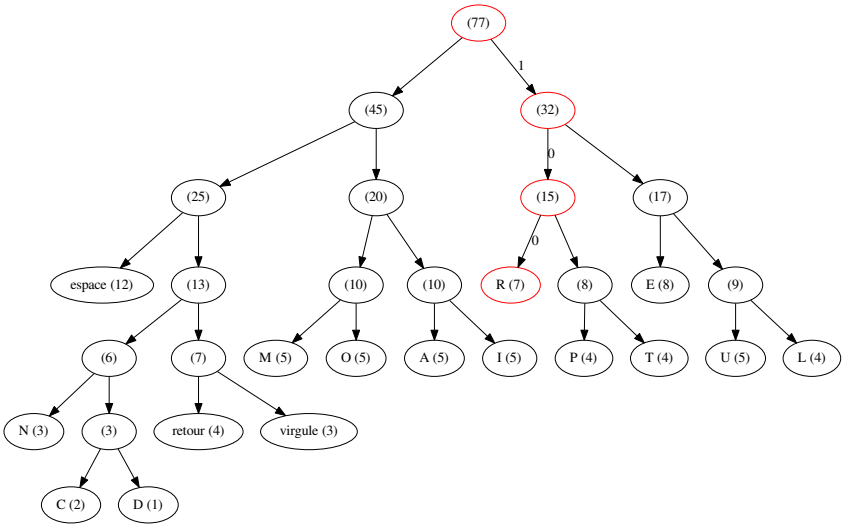


Figure 2: Code de R

(14) Écrivez un programme qui récupère une table des codes et qui décompresse un fichier.