TD n°10-11
Compression par la méthode de Huffman

1 Principes

1.1 Codage de longueur variable

En 1952, **David Huffman** met au point une méthode de compression sans perte basée sur l'étude statistique du document à traiter et sur la construction d'un arbre permettant la construction d'un codage de longueur variable.

L'idée est d'associer aux caractères[1] du document à compresser, une séquence de bits d'autant plus courte que leur fréquence est élevée.

C'est d'ailleurs le principe qui avait été utilisé plus d'un siècle plus tôt pour définir le code Morse (1838, **Samuel Morse** et **Alfred Vail**).

Vous avez dit bizarre devient en morse[2] :

Vous/avez/dit/bizarre/

En utilisant l'arbre de codage Morse présenté dans le poly de cours n°4, coder la phrase suivante :

Hello/IG2I/

1.2 Méthode d'Huffman : principe général

La méthode de compression de **Huffman** procède de la façon suivante :

1. On recherche tout d'abord le nombre d'occurrences de chaque caractère du document à traiter.
2. Puis, on construit l'arbre de codage de **Huffman** en partant des feuilles qui, associées aux caractères, portent comme information leur nombre d'occurrences. On associe ensuite les **deux nœuds de plus faible nombre d'occurrences pour former un nouveau nœud interne** dont la valeur est la somme des valeurs de ses fils. On

[1] Le mot caractère est ici pris au sens large. Il désigne l'élément de base du document traité, l'octet de façon générale.

[2] La convention d'écriture utilisée ci-dessus consiste à séparer chaque caractère d'un mot par un /, et à séparer chaque mot d'une phrase par //.

réitère ce processus avec les feuilles et les nœuds internes restants jusqu'à ne plus en avoir qu'un seul, la racine.

3. L'arbre de codage étant créé, le codage d'un caractère est déterminé par le chemin depuis la racine, permettant d'atteindre la feuille qui lui est associée. À la branche conduisant au fils de plus grand nombre d'occurrences on associe un bit à 1, à l'autre un bit à 0 (ou inversement, car peu importe).
4. On utilise le codage ainsi obtenu pour coder chaque caractère du document à traiter.

1.3 Un exemple étape par étape

La phrase à compresser est la suivante :

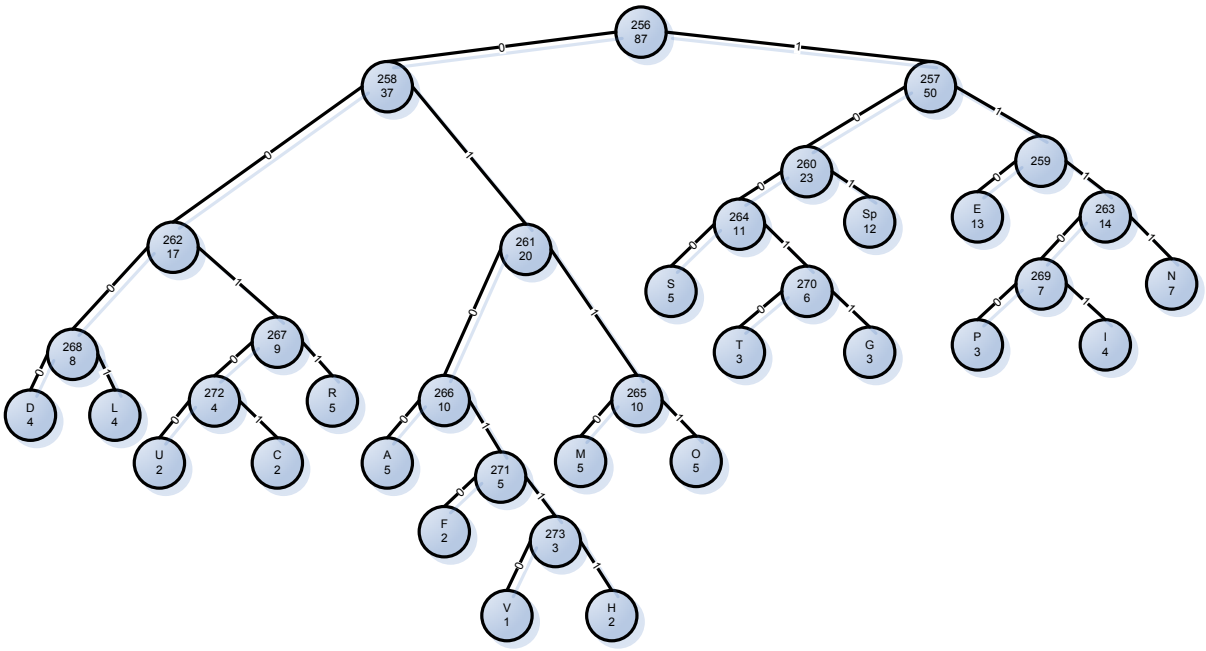
```
IMPLEMENTATION EN LANGAGE C DE L ALGORITHME DE HUFFMAN POUR
COMPRESSER DIVERSES DONNEES
```

La première phase de cette méthode consiste à déterminer le nombre d'occurrences de chaque caractère. On obtient ainsi le résultat suivant :

```
Nombre total de caractères = 87
Nombre de caractères différents = 19
```

occ	Car.
12	' '
5	'A'
2	'C'
4	'D'
13	'E'
2	'F'
3	'G'
2	'H'
4	'I'
4	'L'
5	'M'
7	'N'
5	'O'
3	'P'
5	'R'
5	'S'
3	'T'
2	'U'
1	'V'

On construit ensuite l'arbre de codage :



Enfin, on crée les codes des différents caractères, présents sur les feuilles, à partir des chemins permettant de les atteindre depuis la racine. On obtient ainsi la table de codage suivante :

No	Car.	Code
32	' '	101
65	'A'	0100
67	'C'	00101
68	'D'	0000
69	'E'	110
70	'F'	01010
71	'G'	10011
72	'H'	010111
73	'I'	11101
76	'L'	0001
77	'M'	0110
78	'N'	1111
79	'O'	0111
80	'P'	11100
82	'R'	0011
83	'S'	1000
84	'T'	10010
85	'U'	00100
86	'V'	010110

On peut maintenant utiliser ce codage pour coder le texte initial, ce qui donne la séquence de bits ci-dessous :

```
11101011011100000111001101101111001001001001011101011111111011101
11110100010100111110011010010011110101001011010000110101000110101010
00001100110111001111101100100101110110110101000011010101011100100010
10100101001100100111110111100011100100001110100101011101101110000 1
11101000100011000111010000111010101101100011100011010001010000011 1
111111111101101000
```

Le résultat obtenu est-il satisfaisant ? Comparons pour cela le nombre de bits du document compressé à celui du texte initial codé en ASCII (codage de taille fixe de 8 bits), ainsi qu'à celui que l'obtiendrait avec un codage optimal de taille fixe.

Type de codage	Nombre total de bits	Ratio
ASCII (code de longueur fixe = 8bits)	696	100%
Code de longueur fixe optimal (5 bits)	435	62%
Huffman (code de longueur variable)	348	50%

2 Implémentation

2.1 Représentation de l'arbre d'Huffman

2.1.1 Structure de données & déclarations

L'arbre d'Huffman sera représenté sous la forme d'une table de nœuds. Les feuilles de cet arbre correspondent aux symboles du texte à compresser ; la valeur (le code ASCII) d'un symbole servira alors d'indice dans cette table des nœuds.

Dans l'exemple pris précédemment, les symboles traités sont des caractères et leur code ASCII sert d'indice pour accéder au nombre d'occurrences du caractère et au lien le reliant à son nœud père.

```
typedef struct {
 unsigned int occ;     /* nombre d'occurrences */
 int pere;             /* indice du nœud père  */
} noeud_t;
```

Par convention le fils gauche d'un nœud contient l'indice de ce dernier multiplié par -1. Le signe du champ Père permet ainsi de distinguer le nœud fils droit du nœud fils gauche.

Si *MAX_ELT* représente la taille de l'alphabet initial, c'est-à-dire le nombre maximal de feuilles de l'arbre d'Huffman, alors celui-ci contiendra *MAX_ELT-1* nœuds internes.

Les symboles traités sont des caractères ou de façon plus générale des octets : MAX_ELT vaut donc 256, et **l'arbre d'Huffman sera représenté par une table de 2×*MAX_ELT*-1 nœuds**.

Les feuilles correspondront aux nœuds d'indice 0 à *MAX_ELT*-1, les nœuds internes seront rangés aux indices égaux ou supérieurs à *MAX_ELT*.

```
noeud_t tableNœuds[2*MAX_ELT-1] = { {0, 0} };
```

2.1.2 Exercice : Lecture & Occurrences

Définir les opérations permettant de lire le contenu d'un fichier et d'initialiser cette table de nœuds avec le nombre d'occurrences de chacun de ses éléments

2.2 Création du minimier indirect

2.2.1 Stratégie pour la construction de l'arbre d'Huffman

La stratégie mise en œuvre pour construire l'arbre d'Huffman est dite « gloutonne » :

- On sélectionne les deux nœuds non encore traités ayant le nombre d'occurrences le plus faible.
- On crée alors un nouveau nœud dont le nombre d'occurrences est la somme des nombres d'occurrence des deux nœuds sélectionnés.
- On réitère ces deux étapes jusqu'à ce qu'il ne reste plus qu'un seul nœud non traité qui correspondra à la racine de l'arbre d'Huffman.

Cette stratégie garantit que les feuilles correspondant à des nombres d'occurrences élevés, seront situées à une profondeur inférieure à celle des feuilles de plus faible nombre d'occurrences. On rappelle que la longueur du code affecté aux feuilles correspond à la profondeur de celles-ci.

2.2.2 Quelle mise en œuvre ?

La recherche des deux nœuds de plus faible nombre d'occurrences est une étape clé de cette méthode de compression.

On pouvait envisager de ranger les nœuds dans la table des nœuds, par nombre d'occurrences croissant mais dans ce cas, l'ajout des nœuds internes nécessitera la réorganisation de l'ensemble des nœuds.

Comme il y a *n-1* nœuds internes à ajouter et que les nombres de comparaisons et de déplacements nécessaires à l'insertion d'un nœud sont proportionnels au nombre de nœuds, le coût global de la construction de l'arbre d'Huffman sera en $O(n^2)$.

Une meilleure solution consiste à utiliser un **minimier**. L'accès à l'élément minimal est immédiat, l'ajout ainsi que l'extraction du minimum ont un coût en *log n*. Ce qui conduira à un coût global de construction en $O(n \log n)$.

La table des nœuds ne pouvant pas être réorganisée car **les indices des nœuds servent d'identifiants**, nous utiliserons donc un **minimier indirect** : les éléments de ce minimier sont les indices des nœuds restant à traiter et la clé (de priorité) utilisée par le minimier est le nombre d'occurrences de chaque nœud.

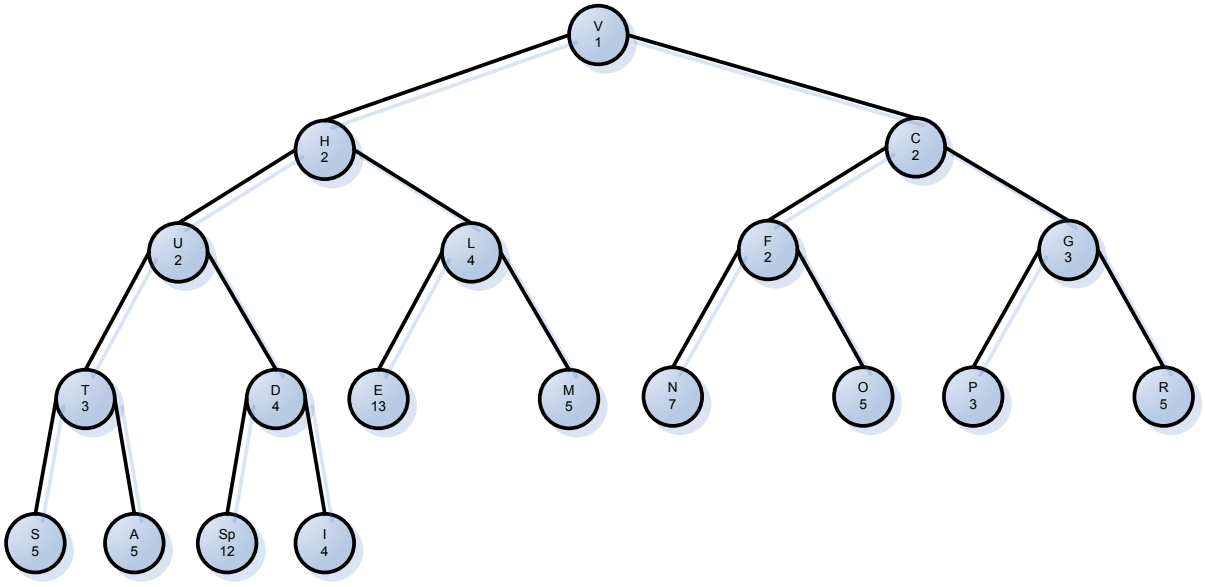
La taille de ce *minimier* sera donc au plus égale au nombre initial de feuilles de l'arbre d'Huffman, c'est-à-dire encore au nombre de caractères différents du document à traiter.

2.2.3 Exercice(s) : Préparation du minimier indirect

- **Donner la déclaration de la structure *minimierIndirect_t* représentant le minimier indirect et d'une variable tas.**
- **Définir le prototype de la fonction *creerMinimier* permettant de construire le minimier indirect initial et retournant la taille du minimier.**
- **Développer la fonction *creerMinimier*. Cette fonction fera appel à une fonction *descendre* similaire à celle présentée lors du TD/cours précédent, utilisant un accès indirect à la clé (occurrence) .**

2.2.4 Retour à notre exemple :

Avec l'exemple pris précédemment, le minimier créé est le suivant :



No	Tas	occ
0	'V'	1
1	'H'	2
2	'C'	2
3	'U'	2
4	'L'	4
5	'F'	2
6	'G'	3
7	'T'	3
8	'D'	4
9	'E'	13
10	'M'	5
11	'N'	7
12	'O'	5
13	'P'	3
14	'R'	5
15	'S'	5
16	'A'	5
17	' '	12
18	'I'	4

2.3 Construction de l'arbre d'Huffman

2.3.1 Principe d'utilisation du minimier indirect pour construire l'arbre de Huffman:

Dans notre exemple, il y a 19 feuilles dans l'arbre d'Huffman et par conséquent 19 éléments dans le *minimier*. Chaque élément de ce dernier est un indice dans la table des nœuds : les indices de 0 à 255 désignent des feuilles et représentent le code ASCII du symbole correspondant.

Les indices supérieurs à 255 désigneront des nœuds internes et contiendront les sommes des nombres d'occurrences de deux nœuds fils.

Dans notre exemple, il y a 18 nœuds internes d'indice 256 à 273 dans la table des nœuds.

```
Creation du noeud 273 valant   3, pere de 'V' et 'H'
Creation du noeud 272 valant   4, pere de 'U' et 'C'
Creation du noeud 271 valant   5, pere de 'F' et 273
Creation du noeud 270 valant   6, pere de 'T' et 'G'
Creation du noeud 269 valant   7, pere de 'P' et 'I'
Creation du noeud 268 valant   8, pere de 'D' et 'L'
Creation du noeud 267 valant   9, pere de 272 et 'R'
Creation du noeud 266 valant  10, pere de 'A' et 271
Creation du noeud 265 valant  10, pere de 'M' et 'O'
Creation du noeud 264 valant  11, pere de 'S' et 270
Creation du noeud 263 valant  14, pere de 269 et 'N'
Creation du noeud 262 valant  17, pere de 268 et 267
Creation du noeud 261 valant  20, pere de 266 et 265
Creation du noeud 260 valant  23, pere de 264 et ' '
```

```
Creation du noeud 259 valant  27, pere de 'E' et 263
Creation du noeud 258 valant  37, pere de 262 et 261
Creation du noeud 257 valant  50, pere de 260 et 259
Creation du noeud 256 valant  87, pere de 258 et 257
```

Lorsqu'un nouveau nœud est créé, il doit être accroché à ses deux nœuds fils, ou plus exactement chaque nœud fils doit être relié à son nœud père.

Le champ Père d'un nœud contient l'indice du nœud père le signe de cet indice indique que le nœud fils est le fils gauche ou le fils droit.

2.3.2 Exercice : construction de l'arbre

Définir les traitements réalisant la phase de construction de l'arbre d'Huffman, et qui permet d'effectuer les affichages présentés ci-dessus.

2.4 Construction de la table de codage

L'avant dernière phase du traitement consiste à construire la table de codage : il suffit de partir des feuilles et de remonter jusqu'à la racine. À chaque remontée d'un niveau on fixe l'un des bits du code d'un caractère à 1 si l'indice de son père est négatif et à 0 sinon.

Nous définirons un code comme une structure à deux composantes :

- Sa taille en nombre de bits
- Sa valeur.

2.4.1 Exercice : Déclaration de la table de codage

On supposera que la taille d'un code est inférieure ou égale à 32.

Définir le type *code_t* et donner la déclaration de la table de codage.

2.4.2 Exercice : génération de la table de codage

Définir les opérations permettant de construire puis d'afficher la table de codage à l'image de l'exemple présenté plus tôt dans cet énoncé.

2.5 Génération du fichier résultat

Lorsque le code de chaque caractère a été obtenu, il ne reste plus qu'à générer pour le document à compresser, le code obtenu pour chacun de ses caractères.

Définir la partie du programme permettant d'afficher sur l'écran le code binaire du fichier compressé.