Projet : Codage de Huffman
Date limite de rendu : le Dimanche 9 Février 2025 à minuit

1 Introduction au codage de Huffman

Le codage de Huffman est une méthode permettant de compresser des fichiers en codant sur peu de bits les symboles fréquents, et en codant sur beaucoup de bits les symboles rares.

Par exemple, imaginons que l'on souhaite coder, de façon la plus compressée possible, la phrase "je regarde le paysage". En temps normal, on associe à chaque caractère sa valeur dans la table ASCII, ce qui nous donne comme code binaire (chaque caractère est codé sur un octet) :

```
01101010 01100101 00100000 01110010 01100101 01100111 01100001 01110010 01100100
01100101 00100000 01101100 01100101 00100000 01110000 01100001 01111001 01110011
01100001 01100111 01100101
```

En tout, notre phrase fait 21 octets, c'est à dire 168 bits. Pour la décoder, un programme utilisera la table ASCII (qui est connue par tous) et associera à chaque octet un symbole. On peut cependant faire mieux en codant sur moins de bits les caractères plus fréquents dans cette phrase.

Par exemple, si l'on choisit ce système de codage :

Espace	j	e	r	g	a	d	l	p	y	s
000	0111	10	0010	0011	010	0110	1100	1101	1110	1111

Alors, notre phrase s'écrira

```
0111 10 000 0010 10 0011 010 0010 0110 10 000 1100 10 000 1101 010 1110 1111 010
0011 10
```

En tout, cette phrase fait 68 bits, c'est à dire 100 bits de moins qu'auparavant. Il faudra, pour décoder cette phrase, transmettre le tableau ci-dessus (ce qui consommera de la mémoire), mais sur les longs textes, ce travail permet d'économiser de la mémoire.

Les codes binaires associés à chaque caractère sont choisis avec prudence, de façon à ne pas créer de confusion lors du décodage de la phrase. Par exemple, dans le codage précédemment donné, si nous décidions de coder l'espace par "00" au lieu de "000". Cela permettrait, en apparence, de diminuer l'espace mémoire consommé par le fichier... Malheureusement, cela entrainerait de la confusion au décodage : la séquence "0010" signifierait-t-elle un espace puis un "e", ou bien un "r" seul ?

L'algorithme de Huffman permet de répondre efficacement à cette problématique.

2 Algorithme de Huffman

L'algorithme de Huffman permet de trouver, étant donné une phrase dans un fichier, une représentation compressée de ces caractères tout en évitant une confusion au décodage. Pour cela, on se propose de construire une structure appelée un "arbre binaire".

Au départ, chaque lettre est représentée par un "noeud", où est stocké la lettre ainsi que le nombre de fois qu'elle apparait dans la phrase ; ce nombre est appelé le poids du noeud. On possèdera ainsi autant de nœuds qu'il y a de lettres différentes dans le fichier. Sur la figure 1, on montre les nœuds obtenus pour l'exemple précédemment cité.

Le but de l'algorithme sera d'assembler les "noeuds" dans un certain ordre, afin de former une structure appelée un arbre binaire.

2.1 Déroulement de l'algorithme

On choisit les deux nœuds de poids les plus faibles, et on créée un nouveau nœud :

— Le poids de ce nouveau noeud sera la somme des poids des deux noeuds précédemment choisis.

— Ce nouveau noeud sera "relié" (à l'aide de pointeurs) aux deux noeuds précédemment choisis. Les deux noeuds choisis seront appelé les "descendants" du nouveau noeud, et le nouveau noeud sera leur "parent". Un noeud parent possède deux descendants, qui seront le descendant droite, et le descendant gauche.

— Les deux nœuds ainsi assemblés ne seront plus pris en compte lorsque l'on cherchera des nœuds à fusionner, mais leur nœud parent le sera.

Sur la figure 2, on montre les nœuds obtenus pour l'exemple précédemment cité.

On recommence la même étape, jusqu'à avoir rassemblés tous les noeuds dans une seule et même structure : on choisit les deux nœuds de poids le plus faible, et on les assemble avec un nœud parent. Sur la figure 3, on montre les nœuds obtenus pour l'exemple précédemment cité à la seconde étape.

On continue l'algorithme, et on note une partie intéressante à l'étape 6, sur la figure 6. A la fin de l'étape 5, les deux noeuds de poids le plus faible sont des noeuds "parents". Ils seront, comme des noeuds ordinaires, assemblés à un même noeud "parent" dont le poids sera 4.

Les figures 1 à 11 montrent toutes les étapes de l'algorithme.

La structure ainsi obtenue est appelée un "arbre binaire". Elle possède un noeud qui ne possède pas de parent (le noeuds de poids 21), que l'on appelle la "racine" de l'arbre. La racine d'un arbre binaire peut être vue comme la tête d'une liste : à partir d'elle, on peut retrouver tous les autres noeuds de l'arbre.

Les noeuds sans aucun descendant sont appelés les "feuilles" de l'arbre, et représentent les lettres de notre texte. Enfin, la structure s'appelle un arbre **binaire** car chaque noeud qui n'est pas une feuille possède exactement deux descendants.

2.2 Comment coder le texte ?

Une fois l'arbre construit, pour savoir comment coder une lettre du texte, on suit le chemin permettant d'aller de la racine de l'arbre à la feuille représentant la lettre. A chaque fois que l'on passe par un descendant gauche, on note un "0", et à chaque fois que l'on passe par un descendant droite, on note un "1".

Par exemple, on voit que, sur l'arbre obtenu à la fin (voir figure 11), pour obtenir la lettre a, il faut partir du nœud racine, descendre à gauche (0), puis à droite (1), puis à gauche (0) : la lettre 'a' sera représentée par le code "010".

3 Que faire ?

Vous devrez proposer deux programmes C permettant respectivement de compresser et de décompresser des fichiers. Vous testerez votre méthode sur les trois fichiers joints au projet, et apporterez vos conclusions sur l'efficacité de la méthode de compression (et tentez d'y apporter des explications). Pour compresser les fichiers images donnés en exemple, considérez-les (et ouvrez-les) comme des fichiers texte que vous lirez caractère par caractère.

Vous devrez rédiger un rapport présentant l'algorithme, votre travail, vos remarques et idées, vos problèmes d'implémentation et vos conclusions.

Voici quelques conseils pour réaliser votre projet :

— Choisissez bien les structures de données qu'il vous faudra pour représenter vos noeuds ainsi que votre arbre. Ce sont des éléments fondamentaux de votre algorithme, et un mauvais choix à ce niveau risque d'entrainer un temps de développement plus long que nécessaire.

— Pensez à ouvrir, dans votre code C, tous vos fichiers avec l'option "b" pour binaire, afin d'éviter que l'OS n'interprète les caractères que vous enverrez comme des commandes et ferme brusquement le fichier. Pour ouvrir un fichier, vous ferez donc :

```
1 FILE *f = fopen("nom_fichier", "rb");
```

— Pensez, lorsque vous compressez votre fichier, à écrire le tableau de décodage des caractères en entête du fichier. En effet, sans ce tableau, impossible de décompresser votre fichier.

— Attention, les fonctions d'écriture et de lecture d'éléments dans les fichiers envoient ou lisent nécessairement au moins un octet dans le fichier... Or, ici, nos éléments sont souvent représentés en moins d'un octet (c'est ainsi que l'on gagne de la place). Comment envoyer des bits (1 ou 0) à des fonctions qui ne prennent que des octets ?

— Etudiez des éléments périphériques au problème, afin d'apporter des questions et des réponses intéressantes dans votre rapport. Par exemple, que donne une double compression des fichiers ? Ou bien, que se passe-t-il si vous ne construisez plus l'arbre en regardant chaque octet (caractère) du fichier, mais en regardant chaque short, int, ou long : les performances de votre compression sont-elles meilleures dans un de ces cas ?

FIGURE 1 – Etape 1 : Formation des nœuds initiaux

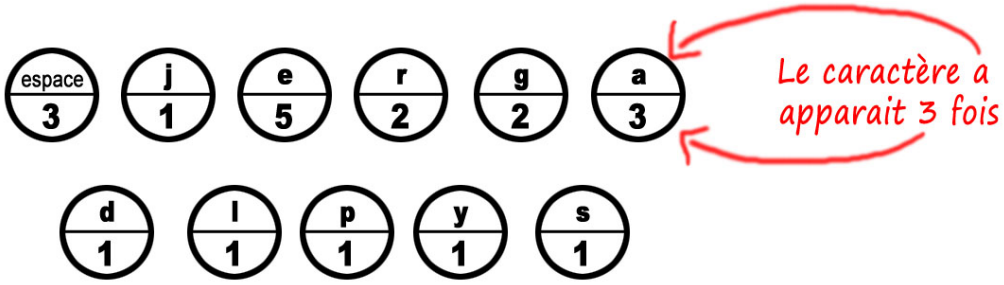


FIGURE 2 – Etape 2 : Première fusion

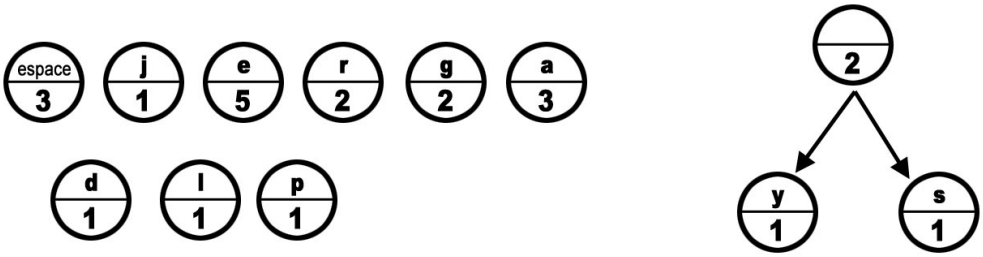


FIGURE 3 – Etape 3 : Seconde fusion

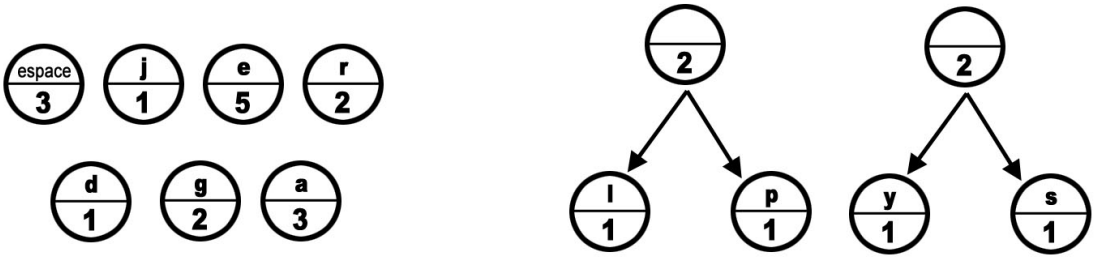


FIGURE 4 – Etape 4

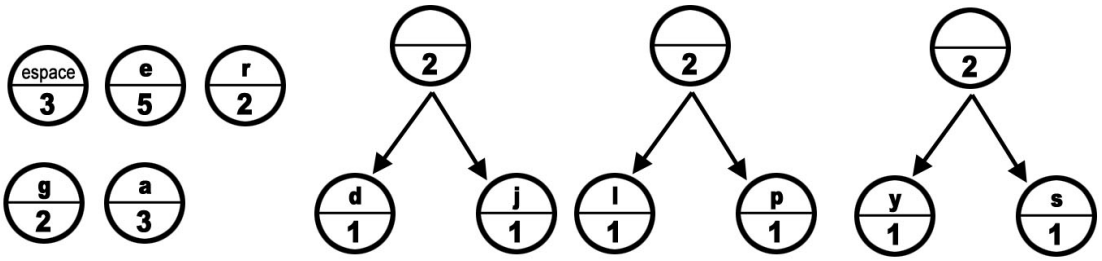


FIGURE 5 – Etape 5

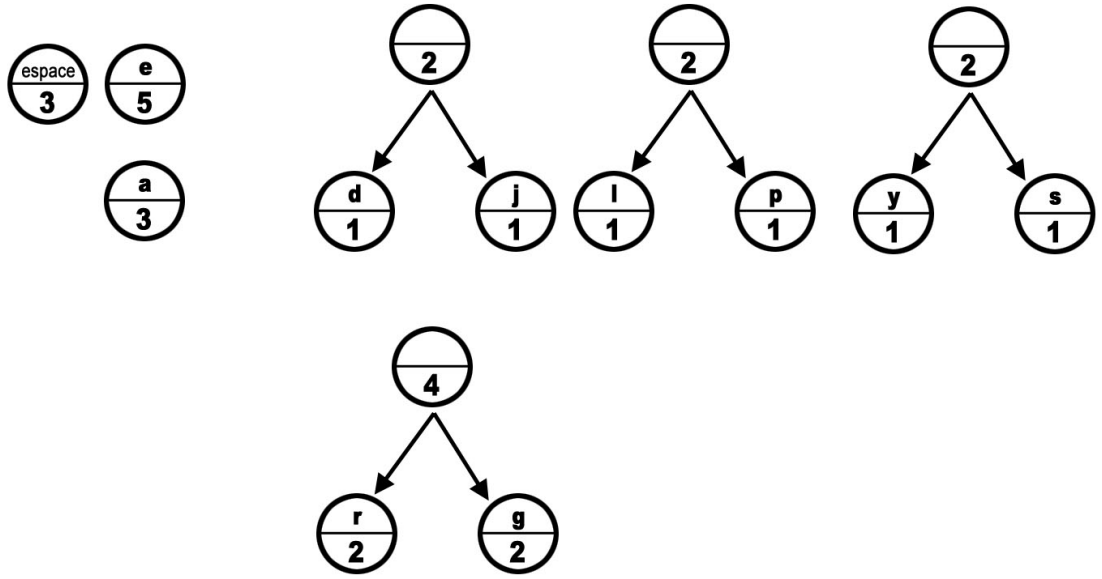


FIGURE 6 – Etape 6

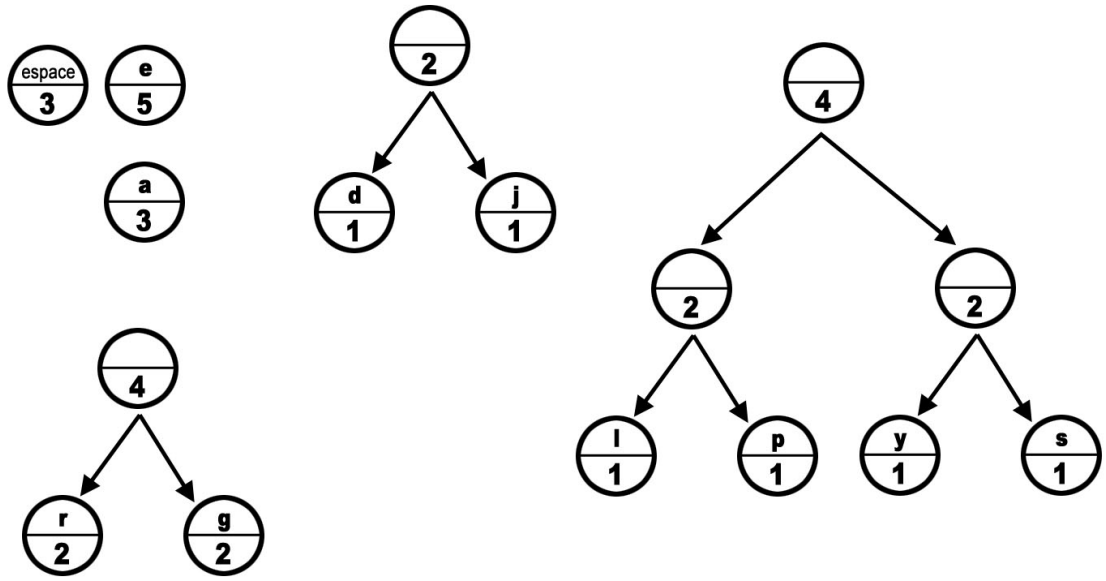


FIGURE 7 – Etape 7

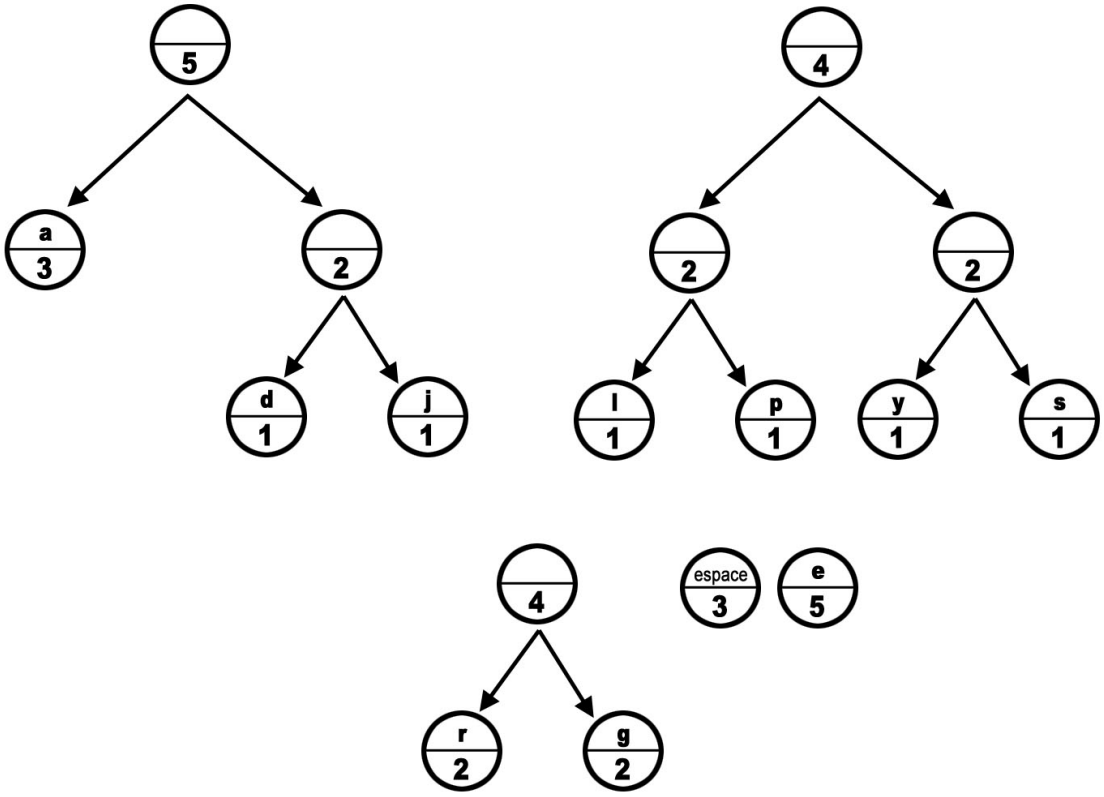


FIGURE 8 – Etape 8

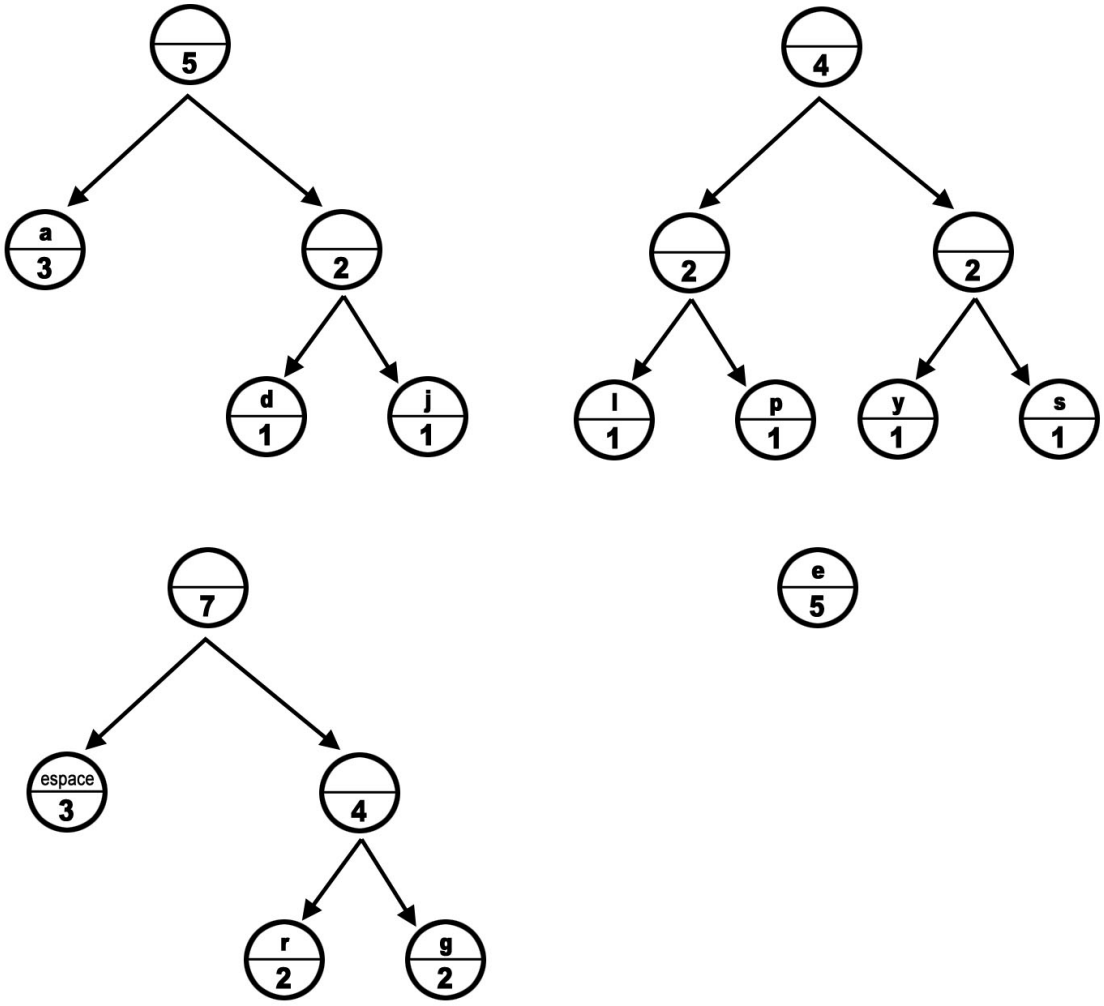


FIGURE 9 – Etape 9

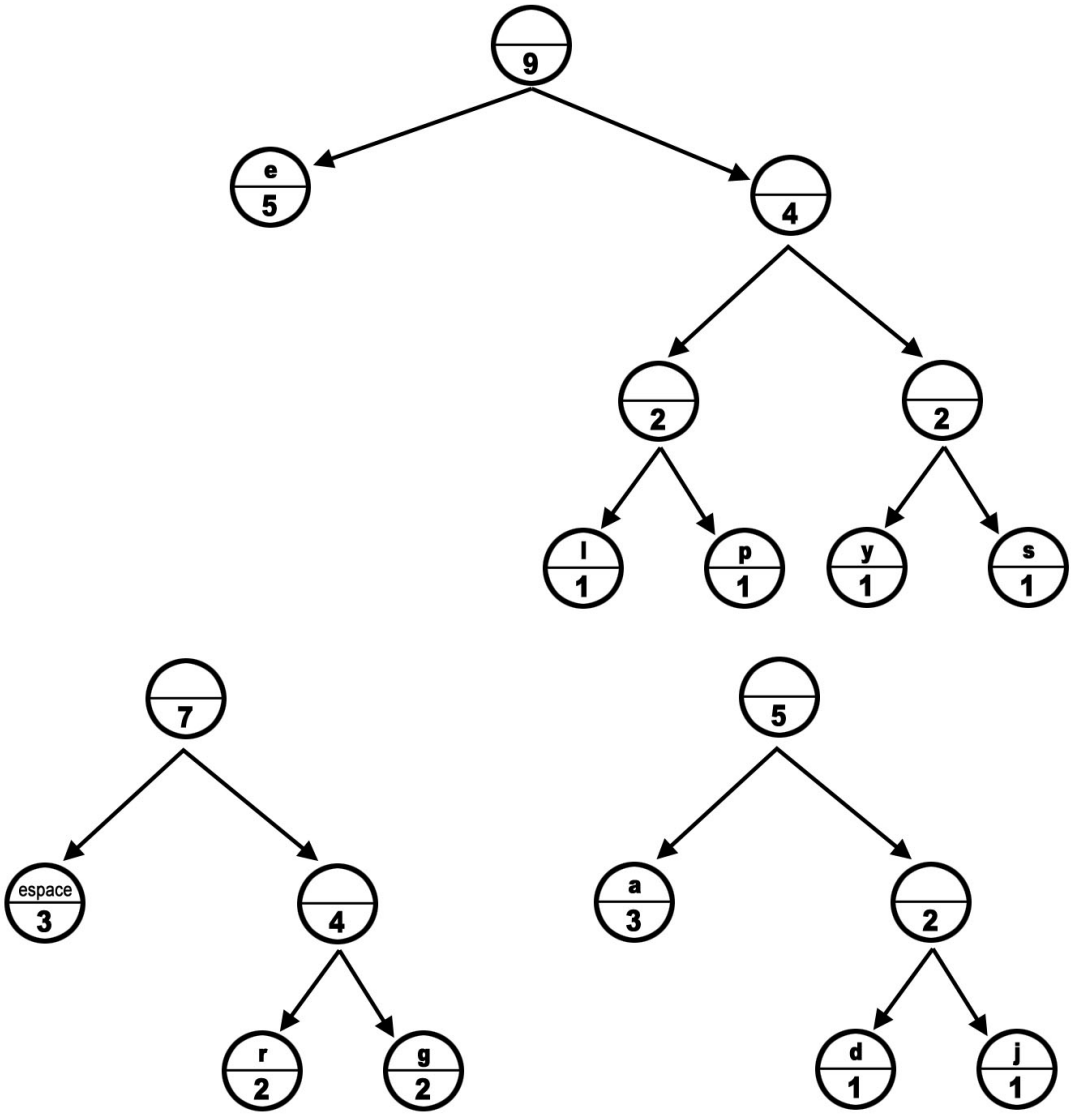
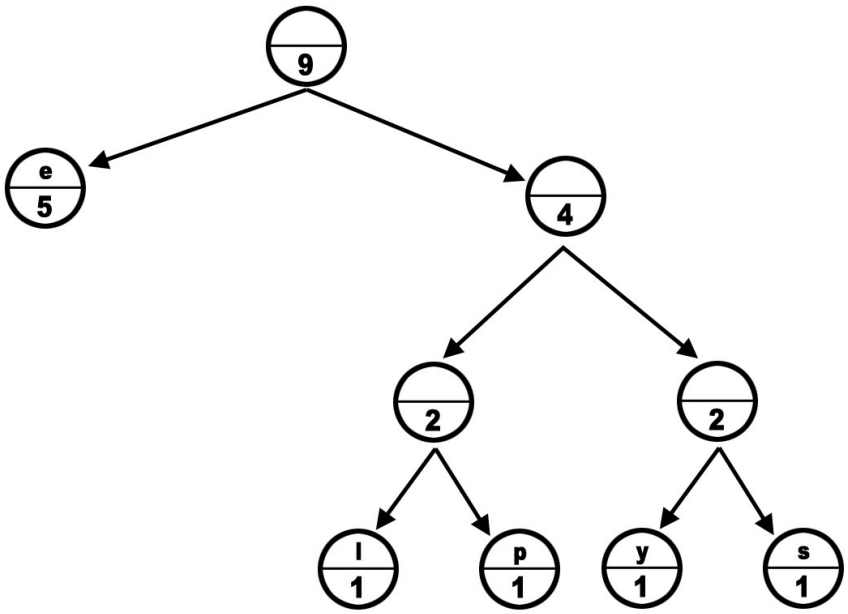


FIGURE 10 – Etape 10



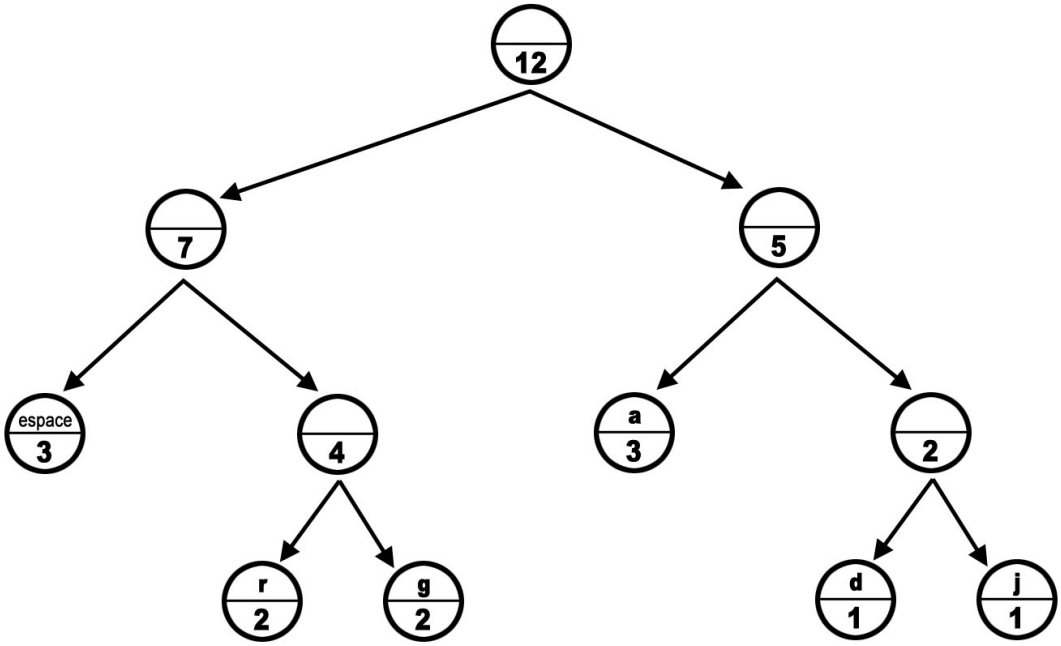


FIGURE 11 – Etape 11