Codage de Huffman

Principe de construction de l'arbre

Dans ce TP on étudie une méthode de compression de données appelée *codage de Huffman* :

Le principe du codage de Huffman repose sur la création d'un arbre composé de nœuds. Supposons que la phrase à coder est " wikipédia ". On recherche tout d'abord le nombre d'occurrences de chaque caractère (ici les caractères 'a', 'd', 'é', 'k', 'p' et 'w' sont représentés chacun une fois et le caractère 'i' trois fois). Chaque caractère constitue une des feuilles de l'arbre à laquelle on associe un poids valant son nombre d'occurrences. Puis l'arbre est créé suivant un principe simple : on associe à chaque fois les deux nœuds de plus faibles poids pour donner un nœud dont le poids équivaut à la somme des poids de ses fils jusqu'à n'en avoir plus qu'un, la racine.

source : *http://fr.wikipedia.org/wiki/Codage_de_Huffman*

Remarque : un arbre de Huffman n'est **pas unique** pour un mot donné.

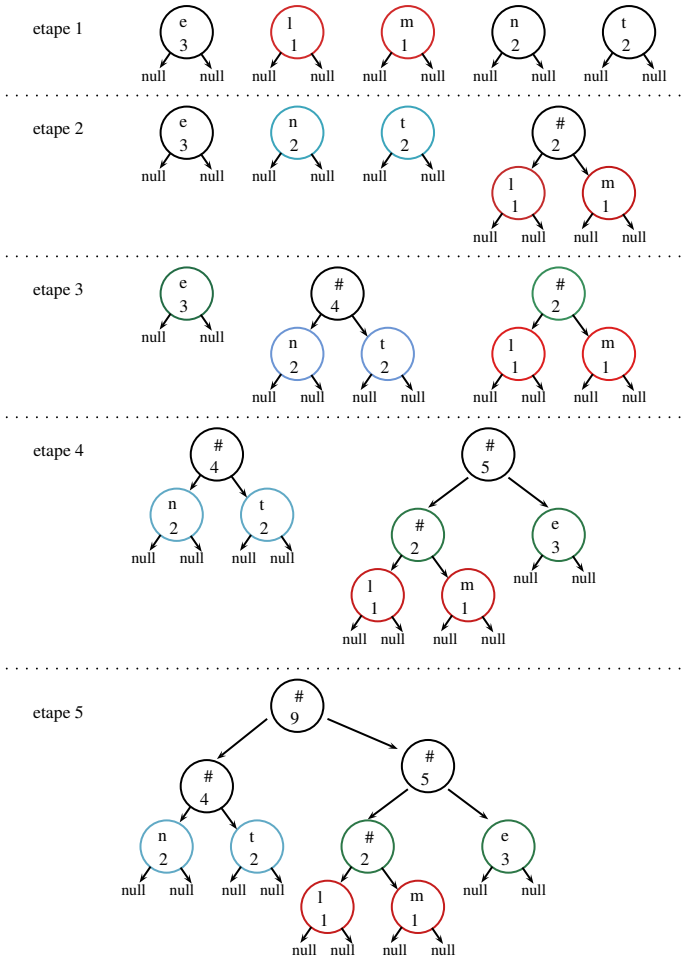


FIGURE 1 – Exemple de déroulement de l'algorithme pour la chaîne de caractères "lentement"

Construction de l'arbre

On considère les 3 classes suivantes :

```
class Arbre {
  private Noeud racine;
  public Arbre(Noeud r){
          this.racine = r;
  }
}
class Noeud {
  private Lettre etiquette;
  private Noeud gauche;
  private Noeud droit;
  public Noeud(char c, int p, Noeud g, Noeud d){
          this.gauche = g;
          this.droit = d;
          this.etiquette = new Lettre(c, p);
  }
  public Noeud(char c, int p){
          this.gauche = null;
          this.droit = null;
          this.etiquette = new Lettre(c, p);
  }
}
class Lettre {
  private char caractere;
  private int poids;
  public Lettre(char c, int p){
          this.caractere = c;
          this.poids = p;
  }
}
```

1. Proposer une fonction `public static int compter(char c, String phrase)` qui dénombre le nombre d'occurrences de la lettre `c` dans la chaîne de caractères `phrase`.

2. Proposer une fonction `public static ArrayList<Noeud> occurrences(String phrase)` qui à partir d'une chaîne de caractères `phrase` construit un `ArrayList` contenant des objets de type `Noeud` correspondants aux différentes lettres présentes dans la phrase (associées à leur occurrence). On suppose que le texte ne contient que des lettres minuscules et des espaces.

3. Écrire la méthode `public static void associer (ArrayList<Noeud> al)` qui à partir de l'`ArrayList al` passé en paramètre trouve les deux nœuds de plus faible poids et les utilise pour former un nouveau noeud qui aura pour fils ces deux noeuds là (avec par exemple le plus petit des deux minimums à gauche), pour attribut `caractere` le caractere '#' et pour attribut `poids` la somme des `poids` de ses fils (voir Figure 1). *(Nb : si l'`ArrayList` possède moins de 2 Noeuds, la méthode ne fait rien.)*

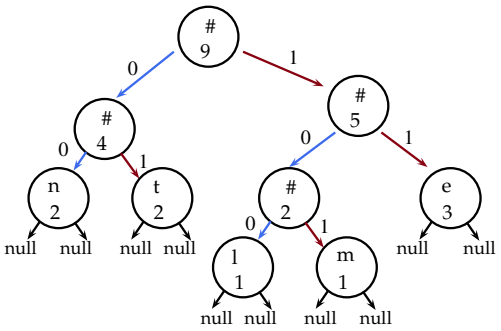
4. Écrire la méthode `public static Arbre construireArbre (String phrase)` qui construit l'arbre tel que décrit dans l'introduction.

5. Tester vos méthodes avec le paramètre `phrase = "lentement"` puis afficher l'arbre produit.

Code Binaire déduit

On associe ensuite le code 0 à la branche de gauche et le code 1 à la branche de droite.
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.

source : http://fr.wikipedia.org/wiki/Codage_de_Huffman



lettre	codage
e	11
l	100
m	101
n	00
t	01

FIGURE 2 – Exemple de code associé pour la chaîne de caractères "lentement"

6. Écrire dans la classe `Arbre` la méthode `public String codage(char c)` qui retourne le codage binaire du caractère `c` donné en paramètre. Si le paramètre n'est pas présent dans this, on retournera la chaîne vide. Tester cette méthode sur l'arbre correspondant à la chaîne de caractères "lentement".

7. Toujours dans la classe `Arbre`, proposer une méthode `public String[] codage()` retournant le codage binaire de toutes les lettres de l'alphabet en utilisant la méthode précédente. Le codage sera retourné sous forme d'un tableau de String contenant les codes de chacune des lettres de l'alphabet dans l'ordre (le codage de 'a' sera à l'indice `0`, celui de 'b' à l'indice `1` et ainsi de suite).

8. Coder une méthode `public static String codage(String phrase)` renvoyant le code de Huffman de `phrase`. Tester la méthode avec la chaîne de caractères "lentement" et d'autres exemples.

9. De façon générale, quel type de lettres est codé sur peu de bits (les plus fréquentes ou les moins fréquentes) ? Comparer la taille d'une phrase quelconque codée avec le code ASCII (voir ci-dessous) et avec votre méthode de codage.

Rappel : L'ASCII est une norme de codage de caractères sur 7 bits permettant de coder les majuscules, les minuscules, les chiffres, des caractères de contrôle ainsi que certains symboles (comme $, %, #). Les lettres minuscules sont codées de 1100001 (soit 97 en décimal) pour 'a' à 1111010 (soit 97+25 = 122 en décimal) pour 'z' :

a	1100001	e	1100101	i	1101001	m	1101101	q	1110001	u	1110101	y	1111001
b	1100010	f	1100110	j	1101010	n	1101110	r	1110010	v	1110110	z	1111010
c	1100011	g	1100111	k	1101011	o	1101111	s	1110011	w	1110111		
d	1100100	h	1101000	l	1101100	p	1110000	t	1110100	x	1111000		

TABLE 1 – Codage ASCII des minuscules