Huffman : compression

- >> ***Compression source.txt destination.bin***
- >> ***Decompression destination.bin source_bis.txt***
- Taille (destination.bin) < Taille (source.txt)
- Taille(source.txt)= Taille(source_bis.txt)
- Utilité : optimiser le stockage et communications
- Huffman est une technique de compression sans perte
- On s’intéresse ici aux fichiers texte.
- Actuellement on s’intéresse plutôt à son et à l’image

Principe

- En principe les caractères dans un fichier sont codés en utilisant leurs codes ASCII : 8bits=1octet
  - ‘a’ ⇒ 97, ‘b’ ⇒ 98, ‘A’ ⇒ 65
- Tous les caractères ont la même taille de code
- Idée de Huffman : distinguer le codage des caractères fréquents de ceux moins fréquents
  - Associer un code très court à un caractère très fréquent
  - Associer un code très court à un caractère très fréquent
  - La taille d’un code d’un caractère augmente avec sa fréquence

Exemple

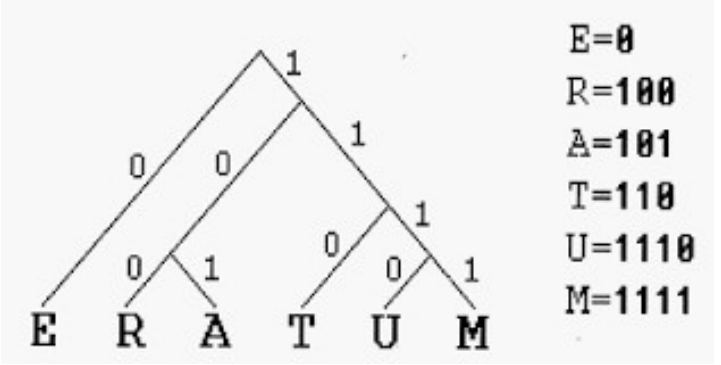
- Soit S=«aaaaaaaaaabbbbbbbbc » la chaîne dont on veut optimiser la codage
- C’est-à-dire minimiser le nombre de bits permettant de la stocker dans un fichier, de la lire à partir d’un fichier

Caractères	a	b	c
effectif	10	8	1
code	00	01	1

  - Le nombre d’octets = 10*2+8*2+1=37 bits
- Voici un nouveau codage : 28 bits

Caractères	a	b	c
effectif	10	8	1
code	1	01	00

- Un point très important : le codage doit être sans ambiguïté ;
  - a:1 et b:11 : 1111 ⇒ aaaa ou bb ou aab ????
- Solution : utilisation d'un arbre binaire :

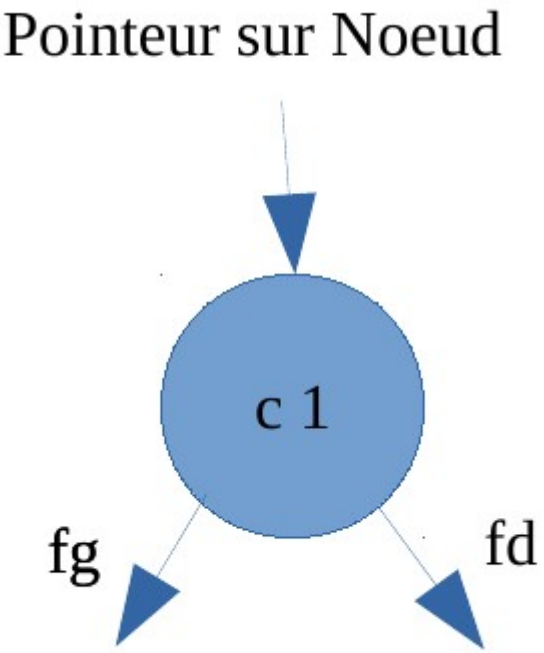


- Aucun code n'est le début d'un autre

Arbre binaire

- La classe Noeud :

```
class Noeud
{
    char c;
    int effectif;
    Noeud *fg;
    Noeud *fd;
    public :
    Noud(char c) ;
    ….
}
```

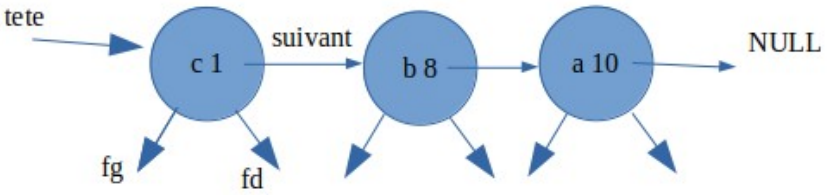


Utilisation d’une liste chaînée

class Noeud

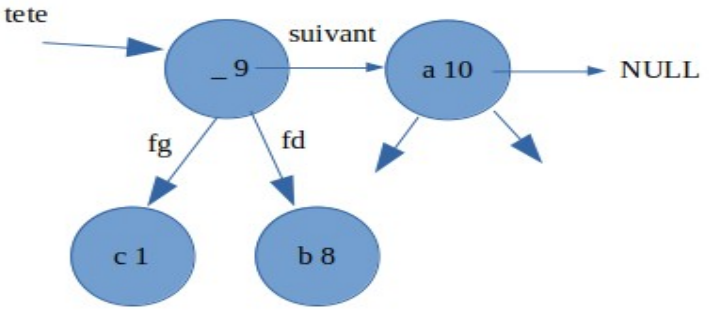
```
{
        friend class arbre ;
        friend classe liste ;

        char c ;
        int effectif ;
        Noeud *fg ;
        Noeud *fd ;
        Noeud *suivant ;
        ….
} ;
```

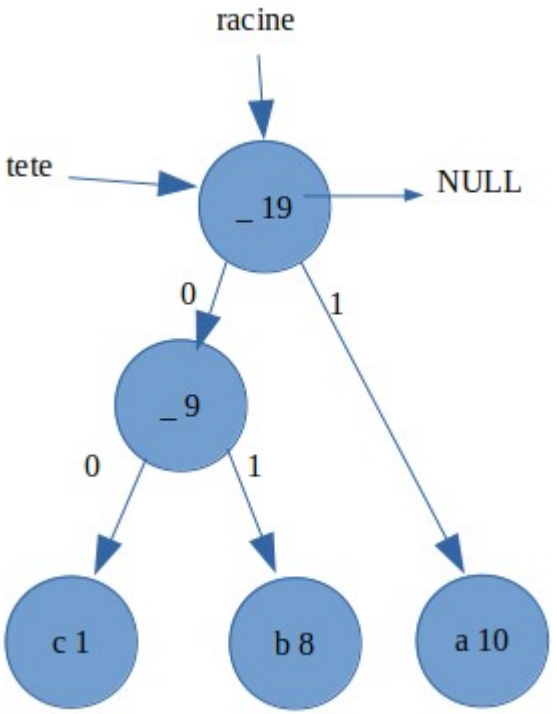


1- Etape 1 : On met tous les caractères dans une u

2- On détermine ensuite les deux Nœuds ayant les effectifs les plus faibles ; ici les Nœuds contenant les caractères « c » et « b ». Ensuite on crée un Nœud dont le fils gauche et droit sont les Nœuds associes à « c » et à « b » : (le nouveau nœud contient la somme des effectifs)



- On refait l’étape 2 tant que la liste contient plus de 1 nœud.



- En parcourant l’arbre de haut en bas, on voit que « c » est codé avec « 00 », « b » est codé avec « 01 » et « a » est codé avec « 1 ».

```
class Noeud
{
     friend class arbre;
     friend class liste ;
     int occ ;
     char c;
     Noeud *suivant ;
     Noeud *fg;
     Noeud *fd;
     public:
     Noeud(const char a_c='\0',int a_occ=0,Noeud *a_suiv,Noeud *a_fg=NULL, Noeud *a_fd=NULL)

} ;

class liste
{
     Noeud *tete;
     int nb_noeuds ;
     public:
     liste() ;
     void inserer_tete(Noeud *n) ;
     void inserer_tete(char c, int effectif) ;
     void inserer_les_caracteres(char *s, int    taille) ;
     Noeud * supprimer_plus_petit() ;
     Noeud * la_tete(){return tete;}
     char *readfile(const string & filename, int & taille) ;
}

class arbre
{
     Noeud *r;
     string codes[256];
     public:
     arbre() { r=NULL ;}
     …..
} ;
```

codes

…….	
a	000
b	0
c	01
d	
……..	

Lecture du fichier

```
char *Liste ::readfile(const string & filename, int & taille)
{
    //création d'un objet de type ifstream : grâce à f on pourra manipuler le fichier
    ifstream f(filename);
    //Aller à la find du fichier
    f.seekg(0,ios::end);
    //taille est le nombre d'octect dans le fichier
    taille=(int)f.tellg()-1;
    //allouer un tableau de la bonne taille
    char *s=new char[taille];
    //se replacer au début du fichier
    f.seekg(0,ios::beg);
    //lire tous les octects du fichier
    f.read(s,taille);
    //fermer le fichier
    f.close();
    //retourner le tableau ; retour de taille par variable
    return s;

}
```

Les fonctions

- void liste ::inserer_les_caracteres(char *s, int taille) ;
  - Pour déterminer les effectifs on crée un tableau d'effectif initialiser à 0 et surtout on accède à l'effectif d'un caractère « c » via son code ASCCI : T[(int)c].
- Noeud* Liste ::supprime_plus_petit() ;
  - qui supprime le Noeud contenant le plus ptit effectif et qui renvoie un pointeur dessus. La fonction renvoie NULL si la liste est vide.
- Noeud *arbre ::construire_arbre(liste & L)
  - Utilise l'algo précédent
- Arbre::arbre(string & name_file)
  - Construit l'arbre de codage
- Void Noeud::visiter(char *, int size, codes)
  - Cette fonction récursive de la classe Noeud permettra en parcourant l'arbre de remplir la table « codes » de la classe arbre.
- Void arbre::codage(){ char code[20] ; int size=0 ; r→visiter(code,size,codes);}
- string & arbre ::codage(char *s, int N, double & taux_compression)