Mini-Projet
Compression

Ce sujet concerne la compression de fichiers texte. L'objectif est de transformer un fichier texte d'une taille N donnée en un autre fichier de taille M avec M<N. Le facteur M/N s'appelle le taux de compression. Le programme doit être réversible, c'est-à-dire qu'on doit pouvoir à partir du fichier compressé retrouver le fichiers d'origine.

>> Compression
Donner le fichier source : source.txt
Donner le fichier destination : destination.bin
Quelle est l'opération : compression
>> Compression
Donner le fichier source : destination.bin
Donner le fichier destination : source_bis.txt
Quelle est l'opération : decompression

Après ces commandes les fichiers source_bis.txt et source.txt doivent être identiques et la taille du fichier destination.bin est inférieure à celle de source.txt.

Méthode :
Idée générale :
Afin d'optimiser globalement le nombre de bits associés aux différents caractères du fichier, on associe les codes les plus courts aux caractères les plus fréquents (un code est une suite de bits).

Exemple :

S=«aaaaaaaaaabbbbbbbbc »

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

Avec le codage ci-dessus, nous arrivons à 37 bits pour coder la séquence de caractères

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

Avec le codage ci-dessus, nous arrivons à 28 bits pour coder la séquence de caractères.

En effet « c » est le caractère le moins fréquent, c'est donc à celui-ci que nous devions associer le code le plus long.

Afin de réaliser cette association caractère/code nous utilisons la notion d'arbre binaire que vous avez vu en cours. Un arbre binaire est un ensemble de nœuds, chaque nœud contient une étiquette (les feuilles contiennent les caractères à coder), un entier correspondant à l'effectif associé au noeud, un pointeur vers un nœud gauche et un pointeur vers un nœud droit.

Tout d’abord on met tous les nœuds correspondants aux caractères de la séquences dans des nœuds feuilles et on les stocke dans un tableau de pointeurs sur Noeuds.

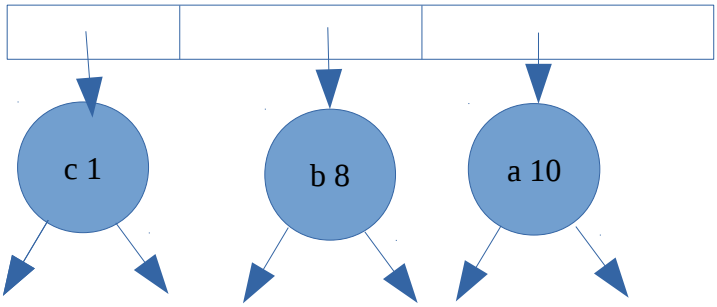
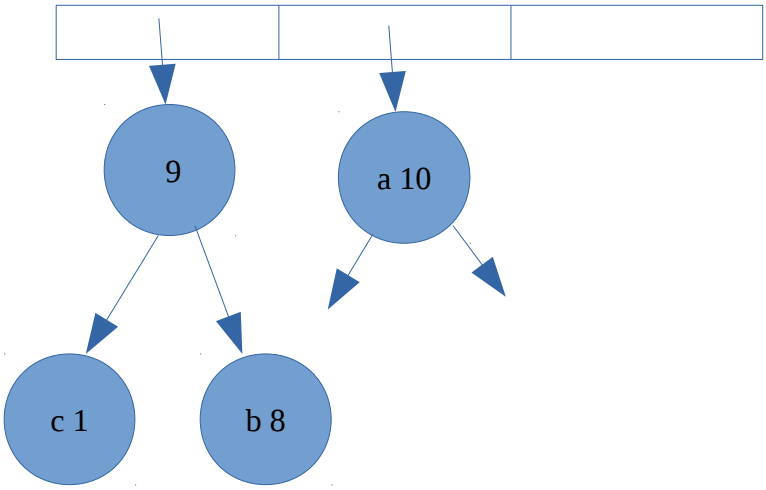
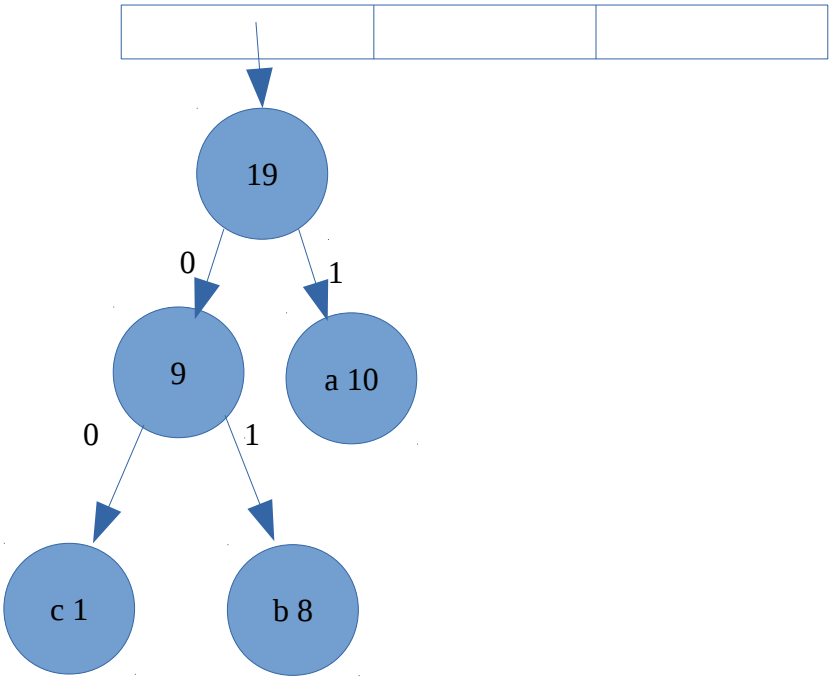


Tableau de pointeurs sur Noeuds

On détermine ensuite les deux caractères ayant les effectifs les plus faibles ; ici 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 » :



Les deux nœuds restant sont deviennent le fils gauche et le fils droit d’un nouveau nœud dont l’effectif est la somme des effectifs.



En parcourant l'arbre de haut en bas, ont voit que « c » est codé avec «00 », b avec « 01 » et a avec « 1 »

Partie I :

Nous souhaitons que toutes les fonctions appartiennent à une classe (pas de fonction en dehors des classes). Pour cela nous créerons une classe appelée ***utilitaires***.

Question 1-

Dans la classe utilitaire, écrire une fonction ***readfile*** qui prend en argument un nom de fichier et qui renvoie un tableau de caractères et un entier représentant la taille de ce tableau.

Question 2-

Dans la classe ***utilitaires***, écrire une fonction ***frequence*** qui reçoit un tableau de caractères et sa taille et qui renvoie un tableau de structures {caractère, effectif}, contenant tous les caractères avec leur effectifs (bien sûr que les effectifs > 0). Considérez que chaque caractère est codé sur un entier compris en 0 et 255 qu'on appelle le code ASCII.

Soit la classe Noeud :

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

Question3-

Ecrire le constructeur de la classe Noeud.

Question 4-

Dans la classe ***utilitaires***, écrire une fonction ***construire_arbre*** qui prend en argument un tableau T de structures (caractère, affectif) et qui renvoie un pointeur sur un nœud. Ce dernier représente la racine de l'arbre de l'algorithme expliqué plus haut, voici les étapes de cette fonction :

> **Etape 1-** Créer un tableau L de pointeurs sur des Noeuds de même taille que T
> **Etape 2-** Pour chaque élément **e** de T, créer un nœud contenant **e** et l'insérer dans le tableau L de façon à ce que L reste toujours trié selon l'ordre croissant des effectifs (utiliser les deux fonction de la correction sur la plate-forme sur les tri : adapter les fonction indice et la fonction inserer_tab_tri).
> **Etape 3-** Appliquer l'algorithme vu plus haut et retourner L[0].

Question 5-
Soit la classe arbre :

```
class arbre
{
    Noeud *racine ;
    public :
       arbre (const string & s) ;
 }
```

Écrire le constructeur qui prend en argument un nom de fichier texte et qui construit l'arbre de codage correspondant (utilisation de la fonction ***readfile, frequence, construire_arbre*** de la classe ***utilitaires***).

Question 6-
Dans la classe Arbre, écrire une fonction (***ecrire***) de sauvegarde d'un arbre de codage en binaire dans un fichier donné. On sauvegarde un nœud avant de sauvegarder son fils gauche puis sont fils droit. N'oublier de sauvegarder s'il s'agit d'une feuille ou pas, c'est important pour pouvoir lire l'arbre à partir du fichier : la fonction ***ecrire*** de la classe arbre fera appel à une fonction récursive de la classe Noeud.

Question 7-
Dans la classe Arbre, écrire une fonction de lecture d'un arbre de codage à partir d'un fichier donné. Cette fonction doit pouvoir reconstituer l'arbre à partir du fichier produit par la question 6. Il s'agit d'une fonction (lire) faisant appel à une fonction récursive de la classe Noeud.

Question 8-
Ajouter à la classe arbre un attribut pour représenter la table de codage :

a	000
b	0
c	01
d	
……..	

Question 9-
Dans la classe Arbre écrire une fonction qui prend en argument un texte (string ou char *) et crée la table de codage (l'arbre est supposé déjà construit). Afin de rendre l'accès un code le plus rapide possible, indexer votre table par le code ASCII du caractère en question.

Question 10-
Écrire une fonction qui prend en argument un texte et une table de codage et qui renvoie un tableau d'entiers constitué de 0 et de 1 représentant la suite de bits de la séquence compressée.

Question 11-
Écrire une fonction qui prend en argument un tableau d'entiers (des 0 et des 1) et qui renvoie un tableau de char contenant les bits correspondant (c'est pas très clair), voici un exemple

T : 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 (16 entiers)
Résultat : 01100010010010010 (2 octects)

Vous devez fournir une implémentation avec les deux méthodes suivantes :

Utilisation de la classe « bitset »

```
#include <iostream>
#include <string>
#include <bitset>
using namespace std;

int main()
{
    bitset<8> X(0);

   X[6]=1 ; //permet de mettre 1 dans le 7 bit partant de la droite (bit de poids faible)
   X[5]=1 ;
   X[0]=1 ;
   char c=X.u_long() ;
   cout<<c<<endl
}
```

Ce petit programme affichera la lettre « a » dont le code ASCII est 97. En fait X contient 01100001=20+25+26=97 qui est bien le code ASCII de « a ».

Utilisation des décalage et des opération bit à bit

```
int main()
{
    char c=0 ;
    int n=1 ;
    for(int i=0;i<8;i++)
    {
       if(i==6 || i==5 || i==0) c=(int)c|(int)n; // avec l'opération | on pose le 1 du n dans c
       n=n<<1 ; // cette opération décale les bits vers la gauche, vers les poids forts
    }
    cout<<c<<endl ;
}
```

Ce petit programme affichera la lettre « a » dont le code ASCII 97 = 01100001

Remarque : si le nombre de bits n'est pas multiple de 8, il faut compléter le dernier avec un code non déchiffrable par l'arbre. Par exemple une partie du plus long code.

Question 12-

Écrire Un programme qui demande à l'utilisateur un nom de fichier source et un nom de fichier destination. Le programme demande ensuite s'il faut procéder en une compression ou en une décompression. Selon le choix de l'utilisateur, le programme doit produire le fichier cible qui soit la version compressée ou décompressée.

Partie II

Nous avons supposé jusqu'à maintenant que les fichiers sont assez petits et leur chargement en mémoire ne pose pas de problème. Ici nous supposons que le fichier est trop gros pour être logé en mémoire en une seule fois. Afin de procéder à sa compression, nous vous proposons de réaliser la compression en deux étapes :

Étape 1

1- lecture bloc par bloc (à vous de définir la taille d'un bloc)
2- cumuler les effectifs provenant des différents blocs (vous pouvez utiliser la fonction frequence)
3- une fois les effectifs calculés sur tout le fichier, construire l'arbre de compression ainsi que la table de codage

Étape 2

1- se remettre au début du fichier, lire bloc par bloc, compresser et stocker dans le fichier destination, n'oublier pas de stocker l'arbre pour la décompression.

Pour la décompression, vous procédez aussi bloc par bloc après avoir construit l'arbre de compression.

Pour des raisons de clarté nous vous demandons d'écrire une version de traitement bloc par bloc indépendante de la version par fichier.

Écrire un programme compression/décompression bloc par bloc.