# Structures de données : arbres binaires

## 1 Arbres binaires

Dans le projet créé pour le TP précédent, ajoutez dans le fichier `main.c` la fonction `test_arbre` suivante :

```
void test_arbre() {
    char rep[8];

    contenu *pc;
    arbre *pa;

    pa=cree_arbre();

    for(;;) {
        printf("\n\
1. (I)nsere arbre\n\
\n\
0. (F)inir\n\
Choix : ");
        saisie_chaine(rep,2);
        switch(rep[0]) {
            case 'I' : case 'i' : case '1' :
                insere_arbre(pa,pc = cree_alea_contenu(0),
                       (int(*)(void *, void *))compare_contenu);

                printf("%-20s : ","Ajout : ");
                affiche_contenu(pc);
                printf("\n");

                printf("%-10s : \n", "Arbree");
                affiche_arbre(pa,(void(*)(void *))affiche_contenu);
                printf("\n");

                break;

            case 'F' : case 'f' : case '0' :
                libere_arbre(pa,(void(*)(void *))libere_contenu);
                mon_free(pa);
                printf("Fin\n");

                return ;
        }
    }
}
```

Ajoutez également les lignes permettant de l'appeler dans le menu du programme principal.

Créez le fichier `arbre.h` contenant les déclarations suivantes :

```
typedef void *arbre;

arbre * cree_arbre() ;
int est_vide_arbre(arbre *a) ;
void insere_arbre(arbre *a, void *pc, int (*compare)(void *, void *)) ;
void libere_arbre(arbre *a, void(* lib_contenu)(void *)) ;
void affiche_arbre(arbre *a, void(* aff_contenu)(void *)) ;

void accroche_gauche(arbre *a, arbre *fg);

void accroche_droite(arbre *a, arbre *fd);

arbre *arbre_gauche(arbre *a);
arbre *arbre_droite(arbre *a);

void *contenu_arbre(arbre * pa) ;
```

Sur le modèle de `liste.c`, créez le fichier `arbre.c`, définissant toutes les structures et fonctions pemettant d'implémenter les fonctions déclarées dans `arbre.h`.

Testez-le et vérifiez que le tri fonctionne bien et que les appels `malloc/free` sont bien appariés.

# 2 Codage de Huffman (2) : le principe du codage

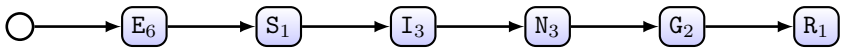
## Présentation

Le codage classique des caractères se fait par une table de correpondance. Avec le codage ASCII, chaque caractère est représenté par un groupe de 8 bits, `01000001` représentant par exemple le caractère `'A'`. Ce codage ne tient pas compte de la fréquence des caractères dans un texte. Par exemple, `'z'` prend autant de place que `'e'`, pourtant bien plus fréquent dans un texte.

Huffman a proposé un codage de longueur variable prenant en compte la fréquence des caractères. Le principe en est le suivant :

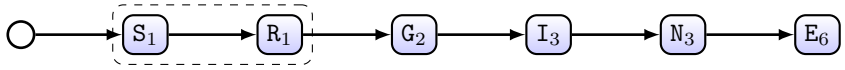
## Construction de l'arbre

- Préparation
  - ▷ À partir de la chaîne initiale, par exemple `"ESIEEENGINEERING"`, on crée la liste des couples (*caractère*, *fréquence*) :

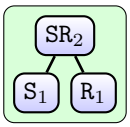



  - ▷ et on la trie par ordre croissant des fréquences :

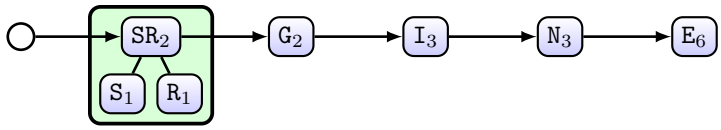
- Étape 1
  - ▷ On sélectionne les deux premiers éléments...



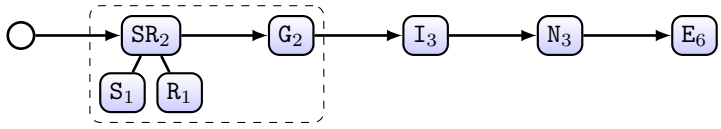

  - ▷ ... que l'on fusionne en un nœud...



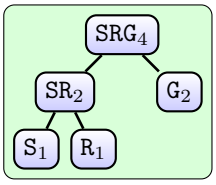

  - ▷ ... que l'on insère à sa place dans la liste.



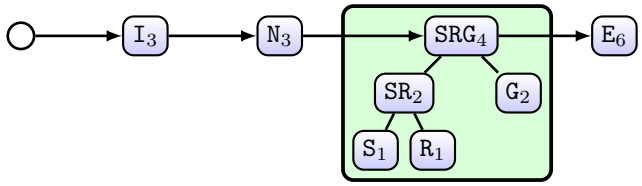

- Étape 2
  - ▷ On sélectionne les deux premiers éléments...



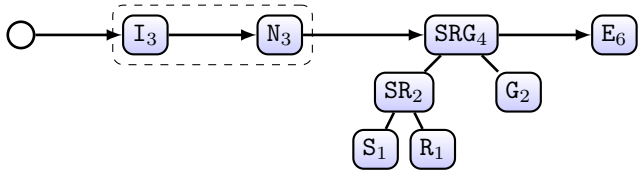

  - ▷ ... que l'on fusionne en un nœud...



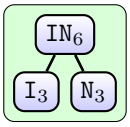

  - ▷ ... que l'on insère à sa place dans la liste.



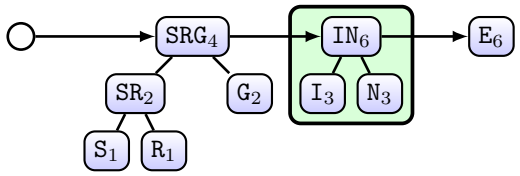

- Étape 3
  - ▷ On sélectionne les deux premiers éléments...



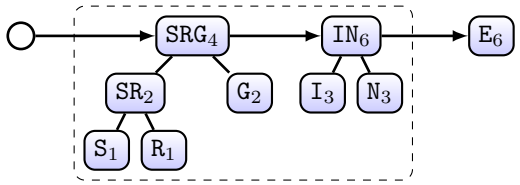

  - ▷ ... que l'on fusionne en un nœud...



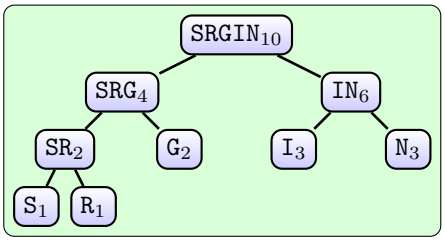
▷ ... que l'on insère à sa place dans la liste.



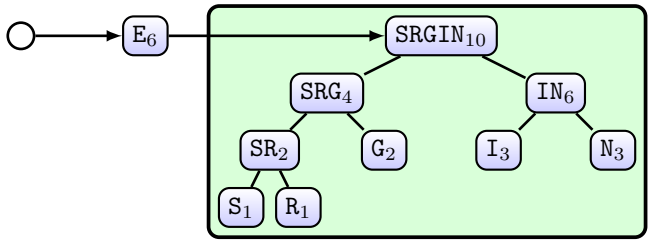

- Étape 4
  - ▷ On sélectionne les deux premiers éléments...



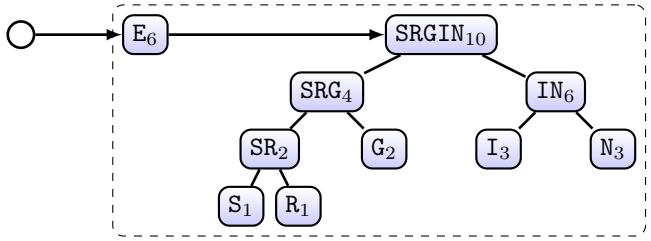

▷ ... que l'on fusionne en un nœud...



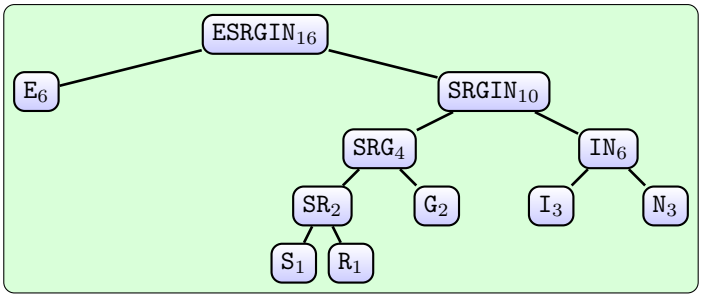

▷ ... que l'on insère à sa place dans la liste.



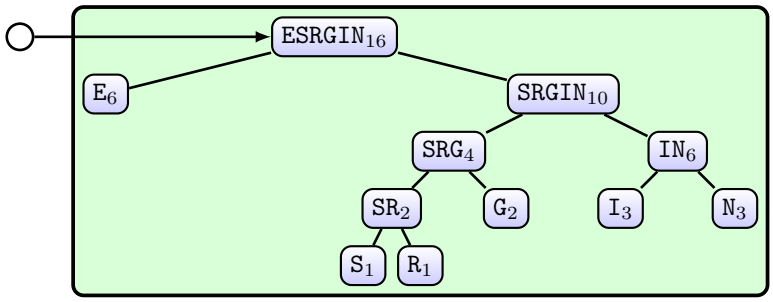

- Étape 5 (la dernière)
  - ▷ Enfin, on sélectionne les deux éléments restants...



▷ ... que l'on fusionne en un nœud...




▷ ... et c'est la liste finale !




## Codage des caractères

Le codage de chaque caractère se fera en cherchant sa position dans l'arbre final en partant de la racine, et en notant 0 si l'on prend à gauchee et 1 si l'on prend à droite.

Ainsi, les codes suivants seront associés aux différents caractères

```
E  0
S  1000
I  110
N  111
G  101
R  1001
```

d'où les codages suivants : `E 0, S 1000, I 110, N 111, G 101, R 1001`

Le codage de ESIEEENGINEERING sera :

```
E  S     I    E  E  E  N    G    I    N    E  E  R     I    N    G
0  1000  110  0  0  0  111  101  110  111  0  0  1001  110  111  101
```

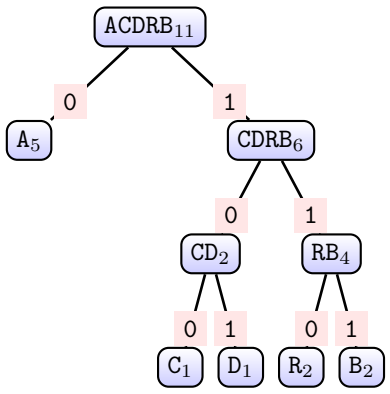
soit 38 bits (le codage ASCII prend lui $16 \times 8 = 128$ bits, et même si l'on codait les cinq caractères sur un codage de largeur constante, il faudrait au minimum 3 bits par caractères, soit 48 bits pour la chaîne). Il faut toutefois noter que quelque part doit être conservé l'arbre ou le moyen de le reconstituer, ce qui prendra une place non négligeable et en tout cas prohibitive pour les petites tailles de données. De plus, l'avantage de la plus faible taille de stockage des données est contrebalancé par le fait que l'exploitation d'un fichier Huffman est impossible en accès direct (on ne peut accéder au 38 178$^{e}$ caractère sans avoir décodé les 38 177 précédents). Il est donc davantage utilisé pour l'archivage des données que pour leur utilisation.

## Décodage

Le décodage se fera de manière inverse. On part de la racine de l'arbre et on lit les bits du codage un par un. Si le bit rencontré est 0, on prend la branche gauche de l'arbre, si le bit est à 1, on prend la branche droite de l'arbre. Si on aboutit à une feuille, on inscrit le caractère et on repart de la racine pour le bit suivant.

Ainsi pour le code `01111100100010101111100` et l'arbre à droite de la figure ci-dessous :

```
Racine
0 : branche gauche A(5) feuille-A
Racine
1 : branche droite CDRB(6)
1 : branche droite RB(4)
1 : branche gauche B(2) feuille -->B
Racine
1 : branche droite CDRB(6)
1 : branche droite RB(4)
0 : branche droite R(2) feuille -->R
Racine
0 : branche gauche A(5) feuille -->A
Racine
1 : branche droite CDRB(6)
0 : branche gauche CD(2)
0 : branche gauche C(1) feuille -->C
Racine
0 : branche gauche A(5) feuille -->A
Racine
1 : branche droite CDRB(6)
0 : branche gauche CD(2)
1 : branche droite D(1) feuille -->D
Racine
0 : branche gauche A(5) feuille -->A
1 : branche droite CDRB(6)
1 : branche droite RB(4)
1 : branche gauche B(2) feuille -->B
Racine
1 : branche droite CDRB(6)
1 : branche droite RB(4)
0 : branche droite R(2) feuille -->R
Racine
0 : branche gauche A(5) feuille -->A
Fin
```




## Travail à faire

Écrivez les fonctions nécessaires à créer à partir d'un tableau de 256 fréquences l'arbre de Huffman correspondant.

Donnez le code de Huffman pour chaque octet présent dans le fichier dico.txt du TP2.