Exercice 1

```
typedef struct Nd_mot_ {
  char *mot;
  struct Nd_mot_ *g;
  struct Nd_mot_ *d;
} Nd_mot;
```

Question 1

```
Lm_mot *part_Lmot(Lm_mot **pl)
{
  int nbm = taille_Lmot(*pl);    déterminer la taille
  Lm_mot *p = *pl;
  Lm_mot *pivot;

  if (nbm < 3)
    {
      *pl = NULL;
      return p;
    }

  nbm /= 2;
  for(p = *pl; nbm--    > 1; p = p->suiv);

  pivot = p->suiv;
  p->suiv = NULL;

  return pivot;
}
```

Question 2

```
Nd_mot *Lm2abr(Lm_mot *l){
  Nd_mot *nd;// pointeur sur noeud
  Lm_mot *p; //pivot


if(l==NULL) return NULL;

nd=malloc(sizeof(Nd_mot));

p=part_Lmot(&l);
nd->mot=p->mot;

nd→d =Lm2abr(p->suiv); fils droite
nd→g  = Lm2abr(l); fils gauche

free(p);

return nd;
}
```

Question 3

```
/* Recherche d'un mot dans ABR */
Nd_mot *chercher_Nd_mot(Nd_mot *abr, const char *mot)
{
  if (abr == NULL) return NULL;

  if (strcmp(mot, abr->mot) == 0) return abr;

  if (strcmp(mot, abr->mot) < 0) return
        chercher_Nd_mot(abr->g, mot);
  return
        chercher_Nd_mot(abr->d, mot);
}
```

Question 4

```
void detruire_abr_mot(Nd_mot *abr)
{
  if (abr == NULL) return;

  detruire_abr_mot(abr->g);
  detruire_abr_mot(abr->d);
  free(abr->mot);
  free(abr);
}
```

Question 5

L’empreinte mémoire d’un élément est de la taille de trois pointeurs (3x (4 octets ou 8 selon l’architecture du pro-
cesseur)) + le nombre de caractères plus un (pour le ’\0’ final). La taille occupée est de (3x8+(m+1))xn, sur une
architecture 64bits, avec n mots de taille moyenne m. Sur une architecture 64 bits, un ABR de 130000 mots d’une
longueur moyenne de 5 lettres occupe donc un espace de
(3x8+(5+1))x130000=3900000o=3808ko=3,7Mo.
Dans ce cas le nombre maximal de comparaisons de mots correspond à la hauteur de l’arbre binaire, 17 pour
130000 mots (17 correspond à log2 (130000)).

Exercice 2

```
typedef struct noeud *PNoeud;
typedef struct noeud {
        char lettre;
        FDM fin_de_mot;
        PNoeud fils;
        PNoeud frere_suivant;
} Noeud;
```

Question 1

```
PNoeud creer_noeud(char lettre){
  PNoeud pn=(PNoeud)malloc(sizeof(Noeud));
  if (pn==NULL) {
    printf("Impossible d'allouer un noeud\n");
    return NULL;
  }

  pn->lettre=lettre;
  pn->fin_de_mot=non_fin;
  pn->frere_suivant=NULL;
  pn->fils=NULL;
  return pn;
}
```

Question 2

```
PNoeud chercher_lettre(PNoeud n, char lettre) {
  if (n==NULL) {
    return NULL;
  }
  if (n->lettre==lettre) {
    return n;
  }
  if (n->lettre>lettre) {
    return NULL;
  }
  return chercher_lettre(n->frere_suivant,lettre);
}



int rechercher_mot(PNoeud dico, char *mot) {

  PNoeud n=chercher_lettre(dico,mot[0]);

  if (n==NULL) {
    return 0;
  }
  if (strlen(mot)==1) {
    return n->fin_de_mot == fin;
  }
  return rechercher_mot(n->fils,mot+1);
}
```

Question 3

PNoeud ajouter_mot(PNoeud racine, char *mot)

chercher le prototype : inserer_lettre

```
void inserer_lettre(PNoeud *racine, PNoeud *n_lettre, char lettre) {
  PNoeud prec=NULL;
  PNoeud n=*racine;

  if (n==NULL) {
    *racine=creer_noeud(lettre);
    *n_lettre=*racine;
    return;
  }

  while(n!=NULL) {
    if (n->lettre == lettre) {
      *n_lettre=n;
      return;
    }

    if (n->lettre>lettre) {
      // on doit inserer avant n
      if (prec==NULL) {
        // insertion en tete
        prec=creer_noeud(lettre);
        prec->frere_suivant=n;
        *racine=prec;
        *n_lettre=*racine;
      }
      else {
        *n_lettre=creer_noeud(lettre);
        prec->frere_suivant=*n_lettre;
```

```
      (*n_lettre)->frere_suivant=n;
     }
     return;
    }
    prec=n;
    n = n->frere_suivant;
  }
  *n_lettre=creer_noeud(lettre);
  prec->frere_suivant=*n_lettre;
}



PNoeud ajouter_mot(PNoeud racine, char *mot) {
  PNoeud n=NULL;
  if (strlen(mot)==0) {
    return NULL;
  }

  inserer_lettre(&racine,&n,mot[0]);

  if (strlen(mot)==1) {
    n->fin_de_mot=fin;
  }
  else {
    n->fils=ajouter_mot(n->fils,mot+1);
  }
  return racine;
}
```

Question 4

Chaque noeud de l'arbre occupe 1+4 (enum) + 2x(4 ou 8). Sur une architecture 64 bits, un noeud occupe donc
21 octets. S'il n'y avait aucune lettre partagée, l'arbre prendrait 21x5x130000=13Mo. Le nombre de noeuds est
en pratique bien inférieur. Dans l'arbre donné en exemple, il y a 24 noeuds pour un total de 34 lettres. Pour
le dictionnaire donné en TME, le nombre de noeuds est de 261.215 (ce qui prend donc 5,2Mo), alors que le
dictionnaire contient 1.270.957 caractères.
Le nombre de comparaisons est très faible par rapport au dictionnaire stocké sous forme d'une liste. Elles se
font lettres par lettres. Dans le pire des cas, il y a 26 comparaisons pour trouver la première lettre (s'il s'agit
d'un 'z'), 26 pour la seconde, etc, soit $26 \times n$ si n est la taille du mot (et si le mot est "zzzzzzzz").