TD9: Arbres et Dictionnaires

Semaine du 26 mars 2007

1 Dictionnaire

▶ **Exercice 1**

```
struct noeud {
        char lettre;
        struct noeud ** fils; /* tableau des fils */
        int n_fils; /* nombre de fils */
};
typedef struct noeud* arbre;
```

▶ **Exercice 2**

```
arbre cree_noeud(char c, int n, arbre * fils){
  arbre a = malloc(sizeof(struct noeud));
  a->lettre = c;
  a->n_fils = n;
  a->fils = fils;
  return a;
}

arbre dico_vide(){
  return cree_noeud('0', 0, NULL);
}
```

▶ **Exercice 3**

```
void detruit_dico(arbre dico){
  int i;
  /* critere d'arret: dico->n_fils == 0 */
  for(i=0;i<dico->n_fils;i++)
    detruit_dico(dico->fils[i]);
  free(dico->fils);
  free(dico);
}
```

▶ **Exercice 4**

```
int nombre_mots(arbre dico){
  int i, nb;
  if (dico->lettre == '\0')
    return 1;
  nb=0;
  for(i=0;i<dico->n_fils;i++)
    nb += nombre_mots(dico->fils[i]);
  return nb;
}
```

▶ **Exercice 5**

```
int nombre_mots_prefixe(arbre dico, char *mot){
  int i;
  while(*mot != '\0'){
    /* dico->fils[i] != NULL car i<dico->n_fils */
    for(i=0; i<dico->n_fils  && (dico->fils[i]->lettre != *mot); i++) {}
    /*le mot n'est pas dans le dico */
    if (i==dico->n_fils)
      return 0;
    dico=dico->fils[i];
    mot++;
  }
  return(nombre_mots(dico));
}
```

▶ **Exercice 6**

```
int recherche_mot(arbre dico, char* mot){
  int i, k, len=strlen(mot);
  for(k=0; k<=len; k++){
    /* dico->fils[i] != NULL car i<dico->n_fils */
    for(i=0;i<dico->n_fils  && (dico->fils[i]->lettre != *mot); i++) {}
    /*le mot n'est pas dans le dico */
    if (i==dico->n_fils)
      return 0;
    dico=dico->fils[i];
    mot++;
  }
  return 1;
}
```

▶ **Exercice 7**

```
arbre nouveau_dico(char *mot){
  arbre nouveau = malloc(sizeof(struct noeud));
  nouveau->lettre = *mot;
  if (*mot == '\0'){
    nouveau->n_fils=0;
    nouveau->fils=NULL;
  }
  else{
    nouveau->n_fils = 1;
    nouveau->fils = malloc(sizeof(arbre));
    nouveau->fils[0] = nouveau_dico(mot+1);
  }
  return nouveau;
}

void ajoute_mot(arbre dico, char* mot){
  int i, k, len=strlen(mot);
  for(k=0; k<=len; k++){
    /* dico->fils[i] != NULL car i<dico->n_fils */
    for(i=0;i<dico->n_fils  && (dico->fils[i]->lettre != *mot); i++) {}
    /*le mot n'est pas encore dans le dico */
    if (i==dico->n_fils) {
      /* realloc(NULL) equivaut a malloc */
      dico->fils = realloc(dico->fils,(i+1)*sizeof(arbre));
```

```
      dico->fils[i]=nouveau_dico(mot);
      dico->n_fils++;
      return;
    }
    dico=dico->fils[i];
    mot++;
  }
}
```

► **Exercice 8**

```
void affiche_dico_rec(arbre dico,char ** mot){
  int i, len;
  if (dico->lettre == '\0') {
    printf("%s\n", *mot);
  } else {
    len = strlen(*mot);
    *mot = realloc(*mot, len + 2);
    (*mot)[len] = dico->lettre;
    (*mot)[len + 1] = '\0';
    for (i = 0; i < dico->n_fils; i++)
        affiche_dico_rec(dico->fils[i], mot);
    *mot = realloc(*mot, len + 1);
    (*mot)[len] = '\0';
  }
}

void affiche_dico(arbre dico){
  int i;
  char *mot=malloc(1);
  *mot='\0';
  for(i=0; i<dico->n_fils; i++)
    affiche_dico_rec(dico->fils[i],&mot);
  free(mot);
}
```

► **Exercice 9**

```
void tri_tab(arbre *tab, int n)
{
  int i,j;
  arbre tmp;
  for (i = 0; i < n; i++)
    for (j = 0; j < n-i-1; j++)
      if (tab[j]->lettre > tab[j+1]->lettre){
        tmp = tab[j];
        tab[j] = tab[j+1];
        tab[j+1] = tmp;
      }
}

void affiche_dico_ordre_rec(arbre dico,char ** mot){
  int i, len;
  if (dico->lettre == '\0') {
    printf("%s\n", *mot);
  } else {
    len = strlen(*mot);
```

```
    *mot = realloc(*mot, len + 2);
    (*mot)[len] = dico->lettre;
    (*mot)[len + 1] = '\0';
    /* on trie le tableau avant d'afficher */
    tri_tab(dico->fils,dico->n_fils);
    for (i = 0; i < dico->n_fils; i++)
        affiche_dico_ordre_rec(dico->fils[i], mot);
    *mot = realloc(*mot, len + 1);
    (*mot)[len] = '\0';
  }
}

void affiche_dico_ordre(arbre dico){
  int i;
  char *mot=malloc(1);
  *mot='\0';
  /* on trie le tableau avant d'afficher */
  tri_tab(dico->fils,dico->n_fils);
  for(i=0; i<dico->n_fils; i++)
    affiche_dico_ordre_rec(dico->fils[i],&mot);
  free(mot);
}
```

► **Exercice 10**

```
arbre supprime_mot_rec(arbre dico, char *mot, int *ok){
  int i;
  if (dico->lettre != *mot){
    *ok = 0;
    return dico;
  }
  if (*mot == '\0'){
    *ok = 1;
    free(dico);
    return NULL;
  }
  for(i=0; i<dico->n_fils; i++){
    dico->fils[i] = supprime_mot_rec(dico->fils[i], mot+1, ok);
    /* derniere bifurcation celle ou il faut tout faire*/
    if (dico->n_fils>1 && *ok){
      echange(dico->fils, i, dico->n_fils-1);
      dico->n_fils--;
      /* realloc(p, 0) equivaut a free(p);*/
      dico->fils=realloc(dico->fils, dico->n_fils);
      *ok=0;
      return dico;
    }
  }
  *ok = (dico->n_fils == 1) && *ok;
  if (*ok){
    free(dico);
    return NULL;
  }
  return dico;
}

void supprime_mot(arbre dico, char *mot){
```

```
  int i, ok;
  for(i=0; i<dico->n_fils; i++)
    dico->fils[i] = supprime_mot_rec(dico->fils[i], mot, &ok);
}
```