TD/TP 13 : arbres préfixes

Gilles Peskine

17 janvier 2003

1 Listes d'association

Une *liste d'association* est une structure de données qui permet d'associer des *données* à des *clés*. C'est une liste de paires, telle que si la donnée d est associée à la clé k, alors la paire (k, d) est dans la liste.

Les opérations fondamentales sur une liste d'association sont :
- créer une liste vide ;
- rechercher la donnée associée à une clé k ;
- ajouter la donnée d associée à la clé k, si k n'est pas déjà présente ;
- supprimer la donnée associée à la clé k ;
- effectuer une action sur chacune des paires (k, d).

Voici une interface des listes d'associations. Les clés sont des entiers ; les données sont des pointeurs génériques (type `void*`).

```
struct cellule;
typedef struct cellule *aliste;

void **aliste_recherche(aliste *l, char c);
void **aliste_ajoute_au_debut(aliste *l, char c, void *d);
int aliste_supprime(aliste *l, char c);
void aliste_parcours(aliste l, void (*f)(char, void**, void*), void* p);
```

Le type d'une cellule de la liste est donné par :

```
struct cellule {
    char c;
    void *donnee;
    struct cellule *s;
};
```

La liste vide est `NULL`.

La fonction `aliste_recherche` recherche dans la liste `*l` une donnée associée à la clé `c`. Elle renvoie l'adresse du champ `donnee` correspondant, afin que l'appelant puisse le modifier. Si la clé est absente, la fonction `aliste_recherche` renvoie `NULL`.

La fonction `ajoute_au_debut` ajoute une paire (clé,valeur) au début de la liste. Une fonction générale d'ajout devrait tester si la clé est déjà présente, mais nous n'aurons besoin que de ce cas particulier.

La fonction `supprime` supprime une clé de la liste. Elle ne fait rien si la clé est absente.

La fonction `aliste_parcours` exécute `f(l->c, &l->donnee, p)` pour toutes les cellules `l` de la liste.

Remarque, pour le projet : il pourra être utile de savoir trier une liste d'association, en ordre croissant des clés.

2 Arbres préfixes

Un arbre préfixe est une structure de données qui permet de stocker un ensemble E de mots, éventuellement en associant une donnée à chaque mot[1]. La particularité des arbres préfixes est que chaque mot est stocké à un nœud n de l'arbre tel que le chemin dans l'arbre de la racine à n soit la liste des caratères du mot.

Voici une interface pour les arbres préfixes.

```
struct ap_noeud;
typedef struct ap_noeud *arbre_prefixe;

arbre_prefixe ap_vide(void);
void ap_insere(arbre_prefixe *a, char* mot, void *donnee);
void ap_supprime(arbre_prefixe *a, char* mot);
void **ap_recherche(arbre_prefixe a, char* mot);
void ap_parcours(arbre_prefixe a, void (*f)(char*, void**, void*), void* p);
```

Le type des nœuds est donné par :

```
struct ap_noeud {
    aliste fils;
    char *mot;
    void *donnee;
};
```

Le champ `fils` est une liste d'association, où les clés sont des caractères et les données des nœuds de l'arbre. Le sous-arbre associé au caractère c contient ceux des mots de l'ensemble E qui commencent par le caractère c.

Le champ `mot` contient le mot correspondant à ce nœud de l'arbre, si le mot en question est présent dans l'ensemble E. Sinon, le champ `mot` contient `NULL`. Si le mot est présent, `donnee` contient la donnée qui lui est associé.

La fonction `ap_insere` ajoute un mot et la donnée associée à un arbre préfixe. La fonction `ap_supprime` supprime un mot (si le mot n'est pas dans l'arbre, elle ne fait rien). La fonction `ap_recherche` recherche un mot, et renvoie l'adresse du champ `donnee` associé si il est présent, ou `NULL` s'il est absent. Enfin, la fonction `ap_parcours` exécute `f(mot, &n->donnee, p)` pour chacun des mots `mot` présents dans l'arbre, `n` étant le nœud correspondant.

[1] Par exemple un nombre d'occurences...