NFP119: feuille d'exercices 5

María-Virginia Aponte

2008

Exercice 1

Considérez le type `'a arbre` étudié en cours. Écrivez les fonctions suivantes.

1. `profondeur` La profondeur d'un arbre est 0 s'il est vide, et égale à la taille (nombre de noeuds) de la branche la plus longue sinon.
2. `existe_arbre` Teste si au moins un des éléments de l'arbre satisfait une condition passée en argument.
3. `map_arbre f a` Renvoie un nouvel arbre obtenu par application de la fonction `f` sur chaque noeud de l'arbre a.
4. `sym_arbre` Renvoie l'arbre symétrique d'un arbre.

Exercice 4

On se donne un type de donnée (voir page suivante) permettant de représenter un dictionnaire (non accentué) sous la forme d'un arbre dont tous les chemins de la racine aux feuilles représentent un mot du dictionnaire. Les nœuds de cet arbre ont un nombre variable de sous-arbres. Si un nœud a zéro sous-arbres, il s'agit d'une feuille. On se donne aussi un dictionnaire de test contenant les chemins suivants : `ARBRE AUX AS BELLE BEC`. Voir la page suivante.

1. Écrire la fonction `dico_assoc` définie comme suit : `dico_assoc c d` rend le `noeud` du dictionnaire `d` ayant le caractère `c` au sommet. On rappelle que la fonction `List.find : ('a -> bool) -> 'a list -> 'a` permet de trouver un élément dans une liste (et lève l'exception `Not_found` en cas d'échec). `dico_assoc` lèvera l'exception `Not_found` en cas d'échec.
2. Écrire la fonction `existe` définie comme suit : `existe d m` rend true si la chaîne de caractère m appartient au dictionnaire d. Il existe plusieurs méthodes : transformation préalable de la chaîne en liste de caractères, utilisation d'un indice pour tester le bon caractère dans la chaîne... On pourra utiliser la fonction `String.get` permettant d'accéder aux $i^{\text{ème}}$ caractère d'une chaîne.
3. Modifier la structure de donnée afin de "marquer" les fins de mots possibles (on n'acceptera pas `ARB`, mais on acceptera `BEL`). Reprogrammez `existe`.
4. Écrire une fonction `ajoute` qui prend une chaîne de caractères et l'ajoute à un dictionnaire.

```
type noeud = Node of char * noeud list
type dico = noeud list

let d7:dico = [Node('t',[])]
let d6:dico = [Node('c',[]);Node('l',[Node('l',[Node('e',[])])])]
let d5:dico = [Node('e',d6)]
let d4:dico = [Node('x',[])]
let d3:dico = [Node('b',[Node('r',[Node('e',[])])])]
let d2:dico = [Node('r',d3@d7);Node('u',d4);Node('s',[])]
let d1:dico = [Node('a',d2);Node('b',d5)]
```

Ce qui donne la valeur suivante pour `d1` :

```
d1=[Node ('a',
     [Node ('r', [Node ('b', [Node ('r', [Node ('e', [])])]);
                  Node ('t', [])]);
      Node ('u', [Node ('x', [])]);
      Node ('s', [])]);
    Node ('b',
     [Node ('e', [Node ('c', []); Node ('l', [Node ('l', [Node ('e', [])])])])])]
```

Et `d1` correspond au schéma suuivant :