1 Liste d'association

1.

```
let rec appartient a = function
  [] -> false
| (x,_)::q -> x=a || appartient a q
;;
```

2.

```
let rec recherche a = function
  [] -> failwith "objet absent"
| (x,b)::q -> if x=a then b
              else recherche a q
;;
```

3.

```
let rec change a b l = function
     [] -> [(a,b)]
   | (x,c)::q -> if x=a then (a,b)::q
                        else (x,c)::(change_rec q)
;;
```

2 Recherche du plus grand préfixe commun

2.1 Conversion chaîne de caractères ↔ liste de caractères

4. On construit la liste dans l'ordre inverse des lettres du mot. Si on parcoure le mot de gauche à droite, il faut utiliser la concaténation de liste qui est plus coûteuse qu'un simple ajout en tête.

```
let char_list_of_string s=
  let n = string_length s in
  let l = ref [] in
  for i=1 to n do
    l:= s.[n-i] :: !l;
  done;
  !l
;;
```

5. Cette solution n'est sûrement pas la moins coûteuse à cause de la concaténation, mais elle est simple à écrire.

```
let rec string_of_char_list = function
    [] -> ""
  | c::q -> (string_of_char c)^(string_of_char_list q)
;;
```

2.2 Utilisation d'un arbre $n-$aire

2.2.1 Recherche dans l'arbre

6.

```
let rec plus_grand_prefixe = function
    Noeud (Mot m,_) -> m
  | Noeud (Prefixe _,[c,a]) -> plus_grand_prefixe a
  | Noeud (Prefixe m,_) -> m
;;
```

On part de la racine de l'arbre et on descent tant qu'il n'y a qu'un seul fils. On s'arrête quand on rencontre un mot, ou bien quand il n'y a plus de fils (une feuille), ou qu'il y en a au moins deux (un embranchement).

7.

```
let rec arbre_prefixe_liste = fun
 | [] a -> a
 | (c::l) (Noeud (e,fils)) -> if (appartient c fils) then
                                 arbre_prefixe_liste l (recherche c fils)
                              else failwith "ce n'est pas un préfixe"
;;
let arbre_prefixe s a = arbre_prefixe_liste (char_list_of_string s) a;;
```

On lit le mot lettre par lettre et on recherche à chaque étape le fils correspondant à la lettre lue. On renvoie l'arbre courant lorsqu'il n'y a plus de lettre à lire.

8.

```
let complete a p= plus_grand_prefixe (arbre_prefixe s p;;
```

On se place d'abord sur l'arbre des mots qui commence par le préfixe `p`, puis on recherche le plus grand préfixe commun de ce sous-arbre.

9.

```
let rec concat_list = function
   [] -> []
  |l::q -> l@(concat_list q)
;;

let rec liste_noms = function
   Noeud (Prefixe _,l) -> concat_list (map (fun (c,a) -> liste_noms a) l)
 | Noeud (Mot m,l) -> m::(concat_list (map (fun (c,a) -> liste_noms a) l))
;;

let trouve_complements a p = liste_noms (arbre_prefixe s p);;
```

La fonction `liste_noms` renvoie la liste des noms présents dans un arbre. La fonction `map` de Caml applique une fonction à tous les éléments d'une liste et renvoie la liste image. Pour la fonction `trouve_complements`, on se place dans le sous-arbre des mots qui commencent par `p` et on appelle `liste_noms`.

2.2.2 Construction de l'arbre

10. Pour alléger le programme, on utilise cette petite fonction qui renvoie la chaîne de caractères d'une étiquette :

```
let nom_etiquette = function
    Mot s -> s
  | Prefixe s -> s
;;
```

Lorsque qu'il n'y a plus de lettres à lire, on transforme l'étiquette courante en un mot : le mot à insérer existait déjà dans l'arbre ou était un préfixe de certains mots de l'arbre.
Sinon, pour chaque lettre du mot à insérer, on regarde si le nœud courant possède déjà une branche étiquettée par cette lettre :
– si oui, on continue l'insertion sur le fils correspondant en modifiant ce fils à l'aide de la fonction `change` et d'un appel récursif,
– si non, on rajoute un nouveau fils au nœud courant avec un appel récursif.

```
let rec insere_mot_liste = fun
  | (Noeud (e,fils)) []   -> Noeud (Mot (nom_etiquette e),fils)
  | (Noeud (e,fils)) (c::l) ->
        if (appartient c fils) then
              let arbre_c = recherche c fils in
              let nouveau_arbre_c = (insere_mot_liste arbre_c l) in
              (Noeud (e,change c nouveau_arbre_c fils))
        else
              let prefixe = (nom_etiquette e)^(string_of_char c) in
              let arbre_c = Noeud (Prefixe prefixe,[]) in
              Noeud  (e,(c,insere_mot_liste arbre_c l)::fils)
;;

let insere_mot a s = insere_mot_liste a (char_list_of_string s);;
```

11.

```
let rec construit_arbre = function
    [] -> Noeud (Prefixe "",[])
  | s::l -> insere_mot (construit_arbre l) s
;;
```