Dictionnaire

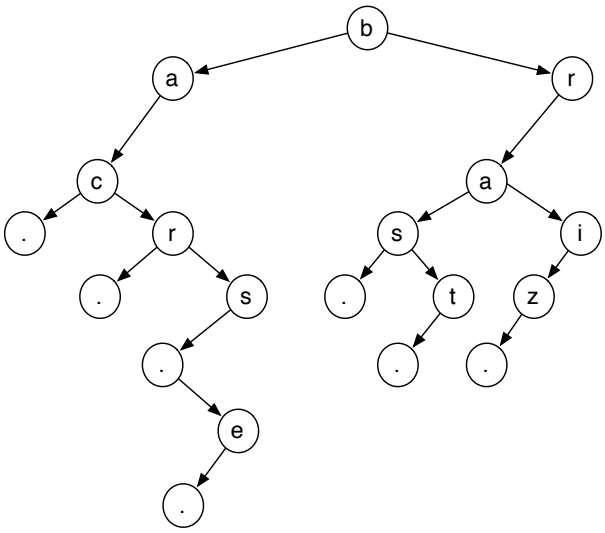
```
let string_of_char = String.make 1 ;;

type dico = Nil | Noeud of char * dico * dico ;;
```

Question 1

Cela permet de déterminer la fin des mots. Sans marqueur, on ne peut distinguer si les préfixes d'un mot sont présents dans le dictionnaire ou non. Par exemple, les mots `base` *et* `bas` *ici sont bien tous les deux dans le dictionnaire.*

Question 2

Il s'agit de décrire le dictionnaire sous la forme d'un arbre binaire.
Le fils droit signifie "frère", et le fils gauche signifie "fils".




```
let d = Noeud
  ('b',
   Noeud
    ('a',
     Noeud
      ('c', Noeud ('.', Nil, Nil),
       Noeud
        ('r', Noeud ('.', Nil, Nil),
         Noeud
          ('s', Noeud ('.', Nil, Noeud ('e', Noeud ('.', Nil, Nil), Nil)),
           Nil))),
     Nil),
   Noeud
    ('r', Noeud ('a', Noeud
        ('s', Noeud ('.', Nil, Nil), Noeud ('t', Noeud ('.', Nil, Nil), Nil)),
      Noeud ('i', Noeud ('z', Noeud ('.', Nil, Nil), Nil), Nil)),
     Nil));;
```

Question 3

```
let chercher m a =
  let n = String.length m in
  let rec aux k b = match b with
    | Nil                              -> false
    | Noeud ('.', _, _) when k = n     -> true
    | Noeud (_, _, a2)  when k = n     -> aux k a2
    | Noeud (c, a1, _) when m.[k] = c -> aux (k+1) a1
    | Noeud (_, _, a2)                 -> aux k a2
  in aux 0 a ;;
```

Question 4

```
let branche m =
  let n = String.length m in
  let rec aux i = match i with
    | k when k = n -> Noeud ('.', Nil, Nil)
    | k            -> Noeud (m.[k], aux (k+1), Nil)
  in aux 0 ;;
```

Question 5

```
let inserer m a =
  let n = String.length m in
  let rec aux k b = match b with
    | Nil                                    -> branche (String.sub m k (n-k))
    | Noeud ('.', a1, a2) as a when k = n -> a
    | Noeud (c, a1, a2) when k = n           -> Noeud (c, a1, aux k a2)
    | Noeud (c, a1, a2) when m.[k] = c       -> Noeud (c, aux (k+1) a1, a2)
    | Noeud (c, a1, a2)                      -> Noeud (c, a1, aux k a2)
  in aux 0 a ;;
```

Question 6

```
let rec creer l = match l with
  | []   -> Nil
  | t::q -> inserer t (creer q) ;;
```

Question 7

```
let extraire a =
  let rec aux acc b = match b with
    | Nil                -> []
    | Noeud ('.', _, a2) -> acc::(aux acc a2)
    | Noeud (c, a1, a2)  -> (aux (acc ^ (string_of_char c)) a1) @ (aux acc a2)
  in aux "" a ;;
```