Exercice 1 [Arbre préfixe]

1. Proposer une structure s'inspirant des arbres de préfixes permettant la définition de dictionnaires (association d'un mot à une définition).

On utilise la structure vue en cours, en remplaçant le booléen par une chaîne de caractères (ou rien si aucune définition n'est associée au mot).

On va utiliser le type `option` : `type 'a option = None | Some of 'a;;`

```
module CharMap = Map.Make(Char);;
type t = T of (string option * t CharMap.t);;

module CharMap = Map.Make(Char);;
type t = (string option * t CharMap.t);; (* problème : récursif *)
```

2. Donner une fonction qui sur la donnée d'une valeur et d'un dictionnaire retourne la liste des clefs correspondant à cette valeur.

```
empty : 'a t
is_empty : 'a t    bool
add : 'a t    char list    'a    'a t
remove : 'a t    char list    'a t
map : 'a t    ('a    'b)    'b t
fold : ('b    'a    'a)    'b t    'a    'a
foldWithKey : (char list    'b    'a    'a)    'b t    'a    'a

type 'a t = T of ('a option * t CharMap.t);;

let f dict v0 =
      foldWithKey (fun k v acc    if v = v0 then k::acc else acc) t [];;
```

Exercice 2 [Graphes]

Dans tout le TD, on représentera les graphes comme vu en cours. On pourra librement utiliser les fonctions définies en cours et aux TD et TP précédents sur ces types.

1. Proposer une fonction permettant de calculer le nombre de noeuds d'un graphe.

```
let nb_noeuds g = G.fold_node (fun _ acc    acc + 1) g 0;;
```

2. Proposer une fonction permettant de calculer le nombre d'arcs d'un graphe.

```
let nb_arcs g = G.fold_node (
      fun n acc    G.NodeSet.fold (fun _ acc'    acc' + 1) (G.succ n) acc
) g 0;;
```

3. Proposer un algorithme permettant d'inverser le sens des arcs d'un graphe. Le graphe obtenu possèdera les mêmes noeuds que G mais ses arcs seront inversés.

Pour tout noeud, on ajoute ce noeud dans le nouveau graphe.
Pour tout arc, on ajoute l'arc opposé dans le nouveau graphe.

```
let inv g =
      let gt0 = G.empty in
      let gt1 = G.fold_node (fun n acc    G.add_node acc n) gt0 g
      in G.fold_edge (fun n n' acc    G.add_edge acc n' n) gt 1 g;;
```

4. Proposer un algorithme permettant de trouver l'ensemble des noeuds accessibles depuis un noeud donné dans un graphe.

```
let rec aux todo seen =
      if (G.NodeSet.is_empty todo) then seen
      else let (n, todo') = G.NodeSet.choose todo in
            let s = G.succ n in
            let seen' = G.NodeSet.add n seen in
            let todo'' = G.NodeSet.fold
                  (fun n' acc    if S.NodeSet.mem n' seen then acc
                                 else S.NodeSet.add n' acc) todo' s
            in aux todo'' seen';;

let accessible g n = aux (G.NodeSet.singleton n)
```