```
(***************************************************************************)
(*           Quelques algorithmes classiques sur les graphes              *)
(***************************************************************************)

type graphe == (int * int list) list ;;

(* Parcours en profondeur *)

let (profondeur : graphe -> int list) = function g ->
  let rec traite_sommet acc = function
    []         -> acc
  | s'::reste -> let acc' = if mem s' acc then
                              acc
                            else
                              traite_sommet (s'::acc) (assoc s' g) in
                 traite_sommet acc' reste

  and traite_graphe acc = function
    []             -> acc
  | (s,al)::reste -> let acc' = if mem s acc then
                                  acc
                                else
                                  traite_sommet (s::acc) al in
                     traite_graphe acc' reste

  in
  rev (traite_graphe [] g)
;;

(* en utilisant it_list : *)

let (profondeur' : graphe -> int list) = function g ->
  let rec traite_sommet a l =
    it_list (fun acc s' -> if mem s' acc then acc else
                           traite_sommet (s'::acc) (assoc s' g)) a l
  and traite_graphe a l =
    it_list (fun acc (s,al) -> if mem s acc then acc else
                               traite_sommet (s::acc) al) a l
  in rev (traite_graphe [] g)
;;

(* Parcours en largeur : on utilise une file (module queue de Caml) *)

let ajoute_liste acc file liste =
  let rec aux = function
      [] -> ()
    | s::reste -> if not (mem s acc) then queue__add s file
  in aux liste
;;

let largeur (g : graphe) =
  let file = queue__new () in
  let rec traite_sommet acc =
    try
      let s = queue__take file in
      if not (mem s acc) then begin
        ajoute_liste (s::acc) file (assoc s g) ;
        traite_sommet (s::acc)
      end else
        traite_sommet acc
    with queue__Empty -> acc
  and traite_graphe acc = function
    []             -> acc
```



```
  | (s,al)::reste ->
      if mem s acc then
        traite_graphe acc reste
      else begin
        queue__clear file ;
        ajoute_liste (s::acc) file al ;
        let acc' = traite_sommet (s::acc) in
        traite_graphe acc' reste
      end
  in
  rev (traite_graphe [] g)
;;

(* Composantes connexes *)

let (composantes : graphe -> int list list) = function g ->
  let rec traite_sommet acc = function
    []         -> acc
  | s'::reste -> let acc' = if mem s' acc then
                              acc
                            else
                              traite_sommet (s'::acc) (assoc s' g) in
                 traite_sommet acc' reste

  and traite_graphe acc = function
    []             -> []
  | (s,al)::reste -> if mem s acc then
                       traite_graphe acc reste
                     else
                       let comp = traite_sommet [s] al in
                       comp::(traite_graphe (comp@acc) reste)

  in
  traite_graphe [] g ;;

(* en utilisant it_list : *)

let (composantes' : graphe -> int list list) = function g ->
  let rec traite_sommet a l =
    it_list (fun acc s' -> if mem s' acc then acc else
                           traite_sommet (s'::acc) (assoc s' g)) a l
  and traite_graphe a l =
    it_list (fun (acc,res) (s,al) ->
                    if mem s acc then acc,res else
                    let comp = traite_sommet [s] al in
                    comp@acc,comp::res) a l
  in snd (traite_graphe ([],[]) g)
;;

(* Recherche d'un cycle *)

let (cyclique : graphe -> bool) = function g ->
  let rec traite_sommet acc orb = function
    []         -> acc
  | s'::reste -> let acc' = if mem s' orb then
                              failwith "cycle"
                            else if mem s' acc then
                              acc
                            else
                              traite_sommet (s'::acc) (s'::orb) (assoc s' g) in
                 traite_sommet acc' orb reste

  and traite_graphe acc = function
```

```
  []             -> false
| (s,al)::reste -> let acc' = if mem s acc then
                                acc
                              else
                                traite_sommet acc [s] al in
                   traite_graphe acc' reste

in
try traite_graphe [] g with Failure "cycle" -> true ;;
```