```
(* question 1 *)

let rec binaire_faible = function
    | 0 -> []
    | n -> (n mod 2)::(binaire_faible (n / 2)) ;;

let binaire_fort n = rev (binaire_faible n) ;;


type ('a, 'b) dfa = {Start: 'a ;
                     Accept: 'a list ;
                     Delta: (('a * 'b) * 'a) list} ;;

exception Not_found ;;


(* question 2 *)

let reconnu a m =
  let rec aux e = function
    | []   -> mem e a.Accept
    | t::q -> aux (assoc (e, t) a.Delta) q
  in try aux a.Start m
    with Not_found -> false ;;


(* question 3 *)

let genere_fort d =
  let rec aux = function
    | q when q = d -> []
    | q -> [(q, 0), (2*q) mod d; (q, 1), (2*q+1) mod d] @ (aux (q+1))
  in {Start = 0; Accept = [0]; Delta = aux 0} ;;

(* test *)
let a = genere_fort 5 in reconnu a (binaire_fort 365) ;;


type ('a, 'b) ndfa = {Nstart: 'a list ;
                      Naccept: 'a list ;
                      Ndelta: (('a * 'b) * 'a) list} ;;


(* question 4 *)

let genere_faible d =
  let a = genere_fort d in
  let rec aux = function
    | [] -> []
    | ((q, a), r)::v -> ((r, a), q)::(aux v)
  in {Nstart = a.Accept; Naccept = [a.Start]; Ndelta = aux a.Delta} ;;


(* question 5 *)

let reconnu2 a m =
  let rec aux e m = function
    | _ when m = [] -> mem e a.Naccept
    | [] -> false
    | ((q, c), r)::v when q = e && c = hd m -> aux r (tl m) a.Ndelta || aux
e m v
    | _::v -> aux e m v
  in exists (function e -> aux e m a.Ndelta) a.Nstart ;;

(* test *)
let a = genere_faible 5 in reconnu2 a (binaire_faible 3664062) ;;
```