```
(***************************************************************************)
(*                          Tables de hachage                              *)
(***************************************************************************)

(* hash m a0a1...an = (a0 + 128.(a1 + 128.(a2 + ... + an)...)) mod m *)

let hash m s =
  let rec hash_rec i res =
    if i = -1 then
      res
    else
      let c = int_of_char (nth_char s i) in
      hash_rec (pred i) ((c + 128*res) mod m)

  in hash_rec (string_length s - 1) 0
;;


type 'a tableh = { M       : int ;
                   donnees : 'a list vect } ;;

let nouvelle m =
  { M = m ;
    donnees = make_vect m [] }
;;

let ajoute t s =
  let i = hash t.M s in
  t.donnees.(i) <- s::t.donnees.(i)
;;

let trouve t s =
  let i = hash t.M s in
  mem s t.donnees.(i)
;;

(***************************************************************************)
(*                   Amelioration : taille dynamique                       *)
(***************************************************************************)

type 'a tableh = { mutable max     : int ;
                   mutable M       : int ;
                   mutable donnees : 'a list vect } ;;

let nouvelle m =
  { max = 3 ;
    M   = m ;
    donnees = make_vect m [] }
;;

let rec trop_long n l =
  if n < 0 then
    true
  else
    match l with
        []    -> false
      | _::l' -> trop_long (pred n) l'
;;

let redim t =
  let new_max = 2 * t.max in
  let new_m   = 2 * t.M + 1 in
  let v = make_vect new_m [] in
```



```
  let rec insert = function
      [] -> ()
    | s::l ->
        insert l ;
        let i = hash new_m s in
        v.(i) <- s :: v.(i)
  in
  for i = 0 to t.M-1 do
    insert t.donnees.(i)
  done ;
  t.max <- new_max ;
  t.M <- new_m ;
  t.donnees <- v
;;

let ajoute t s =
  let i = hash t.M s in
  let l = s :: t.donnees.(i) in
  t.donnees.(i) <- l ;
  if trop_long t.max l then redim t
;;

let trouve t s =
  let i = hash t.M s in
  mem s t.donnees.(i)
;;
```