

Oct 03, 96 8:59

prefixe.ml

Page 1/3

```

(*****
*)          Calcul du plus long complément          (*)
(*****)

type etiquette = Mot      of string
                | Prefixe of string
;;

type arbre = Noeud of etiquette * (char * arbre) list ;;

(* char_list_of_string : string -> char list *)
let char_list_of_string s =
  let rec aux acc = function
    -1 -> acc
    | i -> aux ((nth_char s i)::acc) (pred i)
  in
  aux [] (string_length s - 1)
;;

(* string_of_char_list : char list -> string *)
let string_of_char_list cl =
  let n = list_length cl in
  let s = create_string n in
  let rec aux i = function
    [] -> s
    | c::l -> set_nth_char s i c ; aux (succ i) l
  in
  aux 0 cl
;;

(* change_assoc : ('a * 'b) list -> 'a -> 'b -> ('a * 'b) list *)
let change_assoc l a b =
  let rec change_rec = function
    [] -> []
    | (a',b')::l -> if a=a' then (a',b)::l else (a',b')::(change_rec l)
  in
  if mem_assoc a l then change_rec l else (a,b)::l
;;

(* insere_mot : arbre -> string -> arbre
* Ajoute un mot dans un arbre pour donner un nouvel arbre. *)
let insere_mot arbre mot =
  let rec insere_rec chemin = fun
    (Noeud (Prefixe _,b)) [] -> Noeud (Mot mot,b)
    | a [] -> a
    | (Noeud (e,b)) (c::l) ->
      if mem_assoc c b then
        let fils = insere_rec (c::chemin) (assoc c b) l in
        Noeud (e, change_assoc b c fils)
      else
        let prefixe = string_of_char_list (rev (c::chemin)) in
        let fils = Noeud (Prefixe prefixe,[]) in
        Noeud (e, (c, insere_rec (c::chemin) fils)::b)
  in
  insere_rec [] arbre (char_list_of_string mot)

```

Wednesday April 09, 97

Oct 03, 96 8:59

prefixe.ml

Page

```

;;

(* construit : string list -> arbre
* Construit l'arbre correspondant à une liste de mots *)
let rec construit_arbre = function
  [] -> Noeud (Prefixe "",[])
  | m::l -> insere_mot (construit_arbre l) m
;;

(* Exemple : *)
let a = construit_arbre [ "caml" ; "cafe" ; "cafes" ; "java" ];;

(* descente : arbre -> char list -> arbre
* Descend dans un arbre le long d'une liste de caractères
* et renvoie l'arbre où l'on est arrivé.
* Lève une exception Failure "ce n'est pas un prefixe" si la liste
* ne correspond pas à un chemin dans l'arbre. *)
let rec descente = fun
  a [] -> a
  | (Noeud (e,b)) (c::l) -> if mem_assoc c b then
    descente (assoc c b) l
  else
    failwith "ce n'est pas un prefixe"
;;

(* complete : arbre -> string -> string
* Trouve le plus grand complément dans l'arbre d'un mot donné *)
let complete a mot =
  let rec cherche_complement p = fun
    [c, Noeud (Mot m, b)] -> m
    | [c, Noeud (Prefixe _, b)] -> cherche_complement (c::p) b
    | _ -> string_of_char_list (rev p)
  in
  match descente a (char_list_of_string mot) with
  (Noeud (Prefixe p,b)) -> cherche_complement (rev (char_list_of_string p))
  | (Noeud (Mot m,_)) -> m
;;

(* trouve_mots : arbre -> string list
* Renvoie tous les mots contenus dans l'arbre *)
let applati l = list_it (prefix @) l [];;

let rec trouve_mots = function
  Noeud (Mot m, b) -> m::(applati (map (fun (c,a) -> trouve_mots a) b))
  | Noeud (Prefixe _,b) -> (applati (map (fun (c,a) -> trouve_mots a) b))
;;

(* trouve_complements : arbre -> string -> string list
* Trouve tous les mots de l'arbre dont p est un préfixe. *)
let trouve_complements a mot =
  let a' = descente a (char_list_of_string mot) in
  trouve_mots a'
;;

```

prefixe.ml

```
(*****  
(*                               Optimisation                               *)  
*****)  
  
(* On optimise en notant sur chaque noeud le plus grand prefixe  
* pour un mot aboutissant sur ce noeud-la, au lieu de noter seulement  
* le mot sur lequel on est rendu. *)  
  
let rec optimise a = match a with  
  Noeud (Prefixe p, b) ->  
    let p' = complete a "" in  
    Noeud(Prefixe p', map (fun (c,f) -> c,optimise f) b)  
  | Noeud (Mot m,      b) ->  
    Noeud(Mot m, map (fun (c,f) -> c,optimise f) b)  
;;  
  
let rec complete_opt a mot =  
  match descente a (char_list_of_string mot) with  
  (Noeud (Prefixe m,_) -> m  
  | (Noeud (Mot m    ,_) -> m  
;;
```