Option Informatique
Arbres couvrants minimaux

Corrigé

6 novembre 2006

3 Algorithme de Prim

1.
```
let tri_aretes l_aretes =
  sort_sort
    (fun (Edge(_,_,a)) (Edge(_,_,b)) → a <= b)
    l_aretes
;;
```

2.
```
let poids g =
  let rec aux = function
    | [] → 0
    |(Edge(_,_,p)) : :q → p + aux q
  in aux (g.Aretes)
;;
```

3.
```
random_init (int_of_float (sys_time())) ; ;

let rnd_elt l =
  let rec nth i l = match l,i with
    | _,0 → raise Not_found
    | [],_ → raise Not_found
    | t : :_,1 → t
    | _ : :q,_ → nth (i-1) q
  in nth (random_int (list_length l)+1) l
;;

rnd_elt : 'a list → 'a = <fun>
```

4.
```
let rec cherche_areteP sommets = function
  | [] → failwith "Graphe non connexe"
  | (Edge(x,y,_) as a) : :q →
     let tmpx = mem x sommets and tmpy = mem y sommets in
       if ((tmpx && (not tmpy)) || ((not tmpx) && tmpy))
       then a
       else cherche_areteP sommets q
;;
```

5.
```
let spanningtree_Prim g =
  let rec aux sommets aretes spantree = function
    | [] →  Sommets = sommets ; Aretes = spantree
    | reste →
        let (Edge(x,y,_) as a) = cherche_areteP sommets aretes
        in aux (union sommets [x ;y]) aretes (a : :spantree) (subtract reste [x ;y])
(*
  in let sorted_aretes = tri_aretes g.Aretes
  in match sorted_aretes with
    | [] → failwith "Pas d'arête !"
    | (Edge(x,y,_) as a) : :q → aux [x ;y] q [] (subtract (g.Sommets) [x ;y])
*)
  in let deb = rnd_elt g.Sommets
  in aux [deb] (tri_aretes g.Aretes) [] (subtract (g.Sommets) [deb])
;;
```

6. Il est clair que l'algorithme de Prim termine et fournit un sous-arbre couvrant. Prouvons qu'il est minimal.

   Proof: Soit U une solution minimale telle que le nombre d'étapes de l'algorithme pendant lesquelles l'arbre en cours de construction est un sous-arbre de U soit maximale. Considérons précisement cette étape où l'algorithme rajoute à $F \subset U$ une arête $\alpha = x, y$ qui n'est pas dans U, x est dans le graphe défini par F et non y. Il existe un chemin reliant x à y dans U. Soit β la première arête traversée par ce chemin, qui sort de F. On a $v(\alpha) \leq v(\beta)$ par définition de α dans l'algorithme. Remplaçons β par α dans U pour obtenir U'. U' est encore un arbre-couvrant minimal (supprimer β sépare U en deux composantes connexes que α réunit). Cela contredit la définition de U. □

4 Algorithme de Kruskal

1.
```
let rec fusion foret g1 g2 a =
  let newforet = subtract foret [g1 ; g2]
  in  Sommets = union g1.Sommets g2.Sommets ; Aretes = a : :(union g1.Aretes g2.Aretes) : :new-
foret ; ;
```
2.
```
let explose g = map (function x → Sommets=[x] ; Aretes=[]) g.Sommets ; ;
```
3.
```
let rec cherche_arbre x = function
  | [] → failwith "Boulette !"
  | g : :q →
      if (mem x g.Sommets)
      then g
      else cherche_arbre x q
; ;
```
4.
```
let rec cherche_areteK foret = function
  | [] → failwith "Graphe non connexe"
  | (Edge(x,y,_) as a) : :q →
      let a_x = cherche_arbre x foret and a_y = cherche_arbre y foret in
        if (a_x <> a_y)
        then (a_x, a_y, a)
        else cherche_areteK foret q
; ;
```
5.
```
let spanningtree_Kruskal g =
  let rec aux ssgraphes aretes = match ssgraphes with
    | [] → failwith "Boulette !"
    | [spantree] → spantree
    | _ →
        let (g1, g2, a) = cherche_areteK ssgraphes aretes
        in aux (fusion ssgraphes g1 g2 a) aretes
  in aux (explose g) (tri_aretes g.Aretes) ; ;
```

6. **Proof:** Notons $\alpha = x, y$. $F_1 \backslash \{\alpha\}$ définit deux composantes connexe simples (celle de x, celle de y). Dans le chemin simple reliant x à y dans F_2, on peut choisir une arête β reliant un sommet de la composante connexe de x à un sommet de celle de y. On constate que β convient. □

7. Il est clair que l'algorithme de Kruskal termine et fournit un sous-arbre couvrant. Prouvons qu'il est minimal.

   Proof: Soit F le résultat de l'algorithme et U une solution minimale ayant le plus d'arêtes communes avec F. U et F ont même cardinal. Si $U \neq F$, on prend $\alpha \in U \backslash F$ de telle sorte que $v(\alpha)$ soit minimal. Le lemme donne une arête β de F, $U' = U \backslash \{\alpha\} \cup \{\beta\}$ est un arbre couvrant. Toutes les arêtes de U moins lourdes que α sont dans F par définition de α. La construction de l'algorithme donne alors : $v(\beta) \leq v(\alpha)$ (sinon, α aurait été traitée avant β et aurait été acceptée). Ainsi U' est-il encore minimal, et moins différent de F que U, ce qui contredit sa définition. □