info404 : algorithmes sur les graphes

Complément au TD 5 : chemins de coûtx minimaux

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22
téléphone : 04 79 75 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/

Voila un des deux algorithmes classiques pour calculer les chemins de coûts minimaux à partir d'un unique sommet s dans un graphe sans cycle. (Les poids peuvent être négatifs...) Cet algorithme utilise un tri topologique et est basé sur la remarque suivante : pour tout sommet x, le poids d'un chemin de coût minimal de s vers x est

$$cout(x) \quad = \quad \min\Big(cout(p) + poids(p,x) \;\Big|\; \text{p prédécesseur de } x \Big)$$

Si on prends ceci comme méthode de calcul, il faut être sûr d'avoir calculé le coût optimal pour tous les prédécesseurs de x avant de calculer le coût pour x. L'ordre dans lequel on calcule les coûts doit donc suivre un ordre topologique... De cette manière, on n'a pas a mettre le coût à jours puisqu'on obtient le bon résultat du premier cout !
La fonction utilise les structures de données suivantes :

- `L` : la liste qui contient les sommets dans l'ordre du tri topologique
- `etat[]` : le tableau qui contient l'état (`VU` ou `NON_VU`) de chaque sommet
- `Pred[]` : le tableau des listes des prédécesseurs de chaque sommet
- `cout[]` : le coût d'un chemin de poids minimal pour aller de `s` vers chaque sommet
- `pere[]` : le tableau des pères pour chaque sommet. C'est lui qui permet de récupérer un chemin de coût minimal à partir de `s`

Les trois fonctions importantes sur les listes sont

- `head(L)` qui renvoie la valeur du premier élément de la liste `L`
- `queue(L)` qui renvoie la fin de la liste (toute la liste sauf le premier élément)
- `cons(e,L)` qui renvoie la liste dont la tête est `e` et la queue est `L`

Voir page suivante pour l'algorithme...

```
/*
 * Cette fonction récursive permet de faire un tri topologique (parcours
 * en profondeur) à partir de x et on en profite pour calculer les
 * prédécesseurs de chacun des sommets rencontrés...
 */
fonction tri_top(x) :=
 etat[x] := VU
 pour tous les voisins v de x
 faire
   Pred[v] := cons(x,Pred[v])
   si (etat[v] == NON_VU)
   alors
     tri_top(v)
   finsi
 finfaire
 L := cons(x,L)
finfonction

/*
 * La fonction principale : elle commence par faire le tri topologique et
 * calculer le tableau des prédécesseurs. Ensuite, elle prend tous les
 * sommets dans l'ordre du tri topologique et calcule le coût d'un chemin
 * de poids minimal
 */

/* initialisation */
pour tous les sommets x
faire
  etat[x] := NON_VU
  pere[x] := NULL
  Pred[x] := NILL
finfaire
L := NILL

/* calcul du tri topologique dans L et du tableau des prédécesseurs */
tri_top(s)

/* parcours des sommets dans l'ordre du tri topologique */
cout[s] := 0  ; L := queue(L)    /* on enlève s de la liste... */
tant que L est non vide
faire
  x := tete(L)  ; L := queue(L)
  /* calcul du minimum */
  min := ∞  ; pere := NULL
  pour tous les prédécesseurs p de x
  faire
    si couts[p] + poids(p,x) < min
    alors
     min := couts[p]+poids(p,x)
     pere := p
    finsi
  finfaire
  couts[x] := min
  pere[x] := pere
finfaire
```