info404 : algorithmes sur les graphes

Complément de cours : algorithme de Kruskal

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/

Voilà l'algorithme de Kruskal avec les améliorations vues en cours, et (j'espère) sans erreurs bêtes. Cet algorithme fonctionne sur un graphe représenté par un tableau de listes d'adjacence et calcule un arbre couvrant de poids minimal ainsi que le poids de cet arbre. Il stocke l'arbre en question dans une liste chaînée d'arêtes (liste `T`) et le poids total dans une variable `poids` L'algorithme utilise les variables locales suivantes :

- `L` : la liste de toutes les arêtes
- `CC[]` : un tableau qui contient, pour chaque sommet `x` le numéro de sa composante connexe
- `CCL[]` : un tableau qui contient, pour chaque numéro de composante connexe `i`, une liste chaînée de tous les sommets qui sont dans la composante connexe numéro `i`
- `taille[]` : un tableau qui contient, pour chaque numéro de composante connexe `i`, la taille de la composante connexe numéro `i`

Dans l'algorithme, on suppose que la liste `L` est une liste chaînée où chaque élément `e` contient 3 champs :

- `but` accessible avec "`e->but`"
- `origine` accessible avec `e->origine`"
- `poids` accessible avec "`e->poids`"

Les trois fonctions importantes sur les listes chaînées 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`

Je vous laisse combler les quelques détails manquants comme bon vous semble. (*cf.* vos cours d'algorithmique.)

Voir page suivante pour l'algorithme...

```
/*
 * On commence par trier les arêtes par poids croissant dans une
 * liste chaînée L. Il faut utiliser un algorithme de complexité
 * O(n*log(n)) pour gagner en efficacité.
 * (Par exemple : tri rapide, tri par tas ou tri fusion.)
 */

/* initialisation */
n := "nombre de sommets"
m := 0                  /* nb d'arêtes dans la forêt              */
T := NULL               /* liste chaînée des arêtes dans la forêt */
poids := 0              /* poids total de la forêt                */

pour x de 1 à n
faire
   CC[x] := x               /* au début, chaque sommet            */
   CCL[x] := cons(x, NULL)  /* est tout seul dans sa propre       */
   taille[x] := 1           /* composante connexe                 */
fin faire

/* boucle principale */
tant que m < n-1
faire
  e := tête(L) ; L := queue(L)
  x := e->origine ; y := e->but

  si (CC[x] != CC[y])
  alors
    T := cons(e, T)                /* on rajoute e dans la forêt */
    m := m+1
    poids := poids + e->poids
    si (taille[CC[x]] < taille[CC[y]])
    alors
      i1 := CC[x] ; i2 := CC[y]    /* dans les deux cas,         */
    sinon                          /* i1 contient le numéro de   */
      i1 := CC[y] ; i2 := CC[x]    /* petite composante connexe  */
    finsi

    tant que (CCL[i1] != NULL)
    faire
      z := tête(CCL[i1]) ; CCL[i1] := queue(CCL[i1])
      CCL[i2] := cons(z, CCL[i2])
      CC[z] := i2
    finfaire
    taille[i2] := taille[i2] + taille[i1]
    taille[i1] := 0                /* (facultatif) */)
  finsi
finfaire
```

L'algorithme de Kruskal détaillé...

Question : comment faire pour que l'algorithme mette la valeur `NULL` dans `T` si le graphe n'est pas connexe ?