TD n°8

Tri Topologique – Composantes Fortement Connexes

Exercice 1 [Tri Topologique]

Soit $G = (S, A)$ un graphe orienté ayant n sommets. On interprète un arc (x, y) comme un ordre entre ces sommets : x précède y ou x est avant y. L'objectif du *tri topologique* est de construire une liste de tous les sommets de G tel qu'aucun sommet n'apparaisse avant un de ses prédécesseurs. Un tri topologique est un ordre linéaire des sommets de G. Le tri topologique d'un graphe peut être vu comme un alignement de ses sommets le long d'une ligne horizontale tel que tous les arcs soient orientés de gauche à droite.

Question 1. Effectuer le tri topologique du graphe de la figure 1. Y a-t-il une solution unique ?

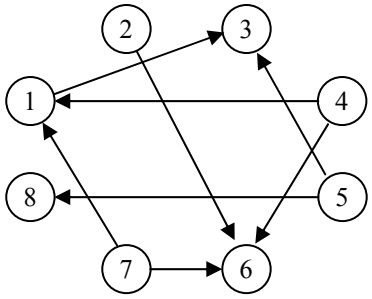


FIGURE 1 – Un graphe orienté

Question 2. Ecrire une fonction `DegreInt` qui renvoie le nombre d'arcs de G ayant pour destination le sommet x (ceci est appelé le degré intérieur).
– dans le cas oú le graphe est représenté par matrice d'adjacence
– dans le cas oú le graphe est représenté par liste d'adjacence
Quelle est sa complexité ?

Question 3. L'idée du tri topologique est de choisir un sommet sans prédécesseur (son degré intérieur est nul), de l'ajouter à la liste résultat et de recommencer avec le graphe privé du sommet que l'on vient de sélectionner.

1. A partir de ceci écrire une fonction récursive utilisant `DegreInt` et `SupprimerSommet`
2. Définir un algorithme itératif, en utilisant un tableau D ayant n éléments tel que :
   $D[s] = -1$ si le sommet d'indice s a déjà été supprimé
   $D[s]$ = le degré intérieur de s mis à jour à partir des sommets déjà supprimés
3. Dérouler l'algorithme à la main.
4. Quelle est sa complexité ? Peut-on faire mieux ?

Question 4. Que se passe-t-il si le graphe contient un cycle ? Modifier l'algorithme pour détecter les cycles dans un graphe.

Exercice 2 [CFC]

Soit $G = (S, A)$ un graphe orienté. On définit sur S la relation R comme suit :

pour tout couple $x, y \in S$: $xRy \iff$ il existe un chemin de x à y et un chemin de y à x

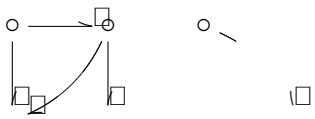
R s'applle relation de $forte\ connexité$

Question 1. Montrer que R est une relation d'équivalence. Donner le nombre maximal et minimal de ses classes d'équivalence.

Soit G/R le graphe $quotient$ qui a pour sommets les classes d'équivalence C_i de R et pour arcs les couples (C_i, C_j) tels qu'il existe un sommet x de C_i et un sommet y de C_j respectivmenet origin et extrémité d'un arc de G.

Question 2. Montrer que G/R est acyclique.

Question 3. Donner le graphe quotient du graphe ci-dessous :



On donne l'algorithme suivant (Algorithme de Kosaraju) :

```
Algorithme de Kosaraju

variables globales:
booleen marquage[V] initialise  a faux
entier  numCC[V]    initialise  a 0
entier  fin[V]
entier  temps = 1

Debut

  // on attribue des numeros suffixes suffixes ...
  Pour chaque sommet x non marque faire
       parcourssuffixe(x)
  Fin pour

  inverser chaque arc de G

  // on diffuse les numeros de CC ...
  i = 1
  Tant qu'il existe un sommet x de numCC[x] == 0 faire
     Soit x le sommet de plus grand fin[] tel que numCC[x] == 0
     diffuser(x,i)
     i = i + 1
  Fin tant que

  inverser chaque arc de G
Fin

Fonction locale parcourssuffixe(sommet x)
Debut
 marquage[x] = vrai
 Pour chaque voisin sortant non marque y de x faire
   parcourssuffixe(y)
 Fin pour
 fin[x] = temps
 temps = temps + 1
Fin

Fonction locale diffuser(sommet x, entier num)
Debut
 numCC[x] = num
 Pour chaque voisin sortant y de x tel que numCC[y] == 0 faire
   diffuser(y,num)
 Fin pour
Fin
```

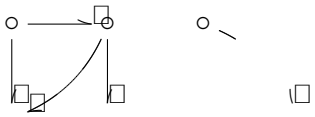


FIGURE 2 – Graphe orientés

Question 4. Exécuter l'algorithme de Kosaraju sur les graphes de la figure 2

Question 5. Expliquer comment l'algorithme de Kosaraju donne les composantes fortement connexes d'un graphe.

Question 6. On veut s'intéresser maintenant à l'implémentation. Expliquer comment faire e□cacement :

```
1.  Pour chaque sommet x non marque faire
2.  inverser chaque arc de G
3.  Soit x le sommet de plus grand fin[] tel que numCC[x] == 0
```

Pour 1. il faut une boucle qui tourne $O(n)$ fois en trouvant à chaque fois le sommet non marqué en temps constant.

Pour 2. on a une enveloppe de temps $O(n + m)$

Pour 3. on peut trouver en temps constant le sommet x, quitte à modifier un peu l'algorithme.

Sous réserve que l'on implémentte bien ces trois points, l'algorithme est en temps linéaire (s'en convaincre).