université de BORDEAUX

Année 2017-2018 — Session de décembre 2017
Étape : INF501-MI501 — UE : 4TIN501U
Épreuve de : Algorithmique des graphes
Date : 14 décembre 2017 - 9h00 — **Durée :** 1h30
Épreuve de : F. Kardoš — **Nb. pages : 8**
Collège Sciences et technologies

Aucun document autorisé. Une attention toute particulière sera portée à la présentation et à la rédaction de la copie.

Dans tous les exercices, on désignera par $V(G)$ et $E(G)$ respectivement l'ensemble des sommets et l'ensemble des arêtes (des arcs) d'un graphe (orienté) G. Les variables n et m désigneront respectivement le nombre de sommets et d'arêtes (d'arcs).

1 Composantes fortement connexes

Soit G un graphe orienté dont la matrice d'adjacence est la suivante :

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

1.1) Appliquer un algorithme de cours adapté pour calculer les composantes fortement connexes de G. Expliciter l'ordre des sommets utilisé pour le deuxième parcours.

1.2) Dessiner G et ses composantes fortement connexes.

1.3) Est-ce possible qu'en ajoutant un seul arc le graphe G devienne fortement connexe ? Justifier.

2 Arbre couvrant de poids minimum

Soit $G = (\{a, b, c, d, e, f, g, h\}, \{ab, ad, bc, bd, be, ce, de, df, dg, eg, eh, fg, gh\})$ un graphe non-orienté.

2.1) Dessiner G. Quel est le degré minimum et degré maximum de G ?

2.2) Dans le tableau ci-dessous se trouvent les poids des arêtes de G. Choisir un algorithme (soit Kruskal soit Prim) et calculer un arbre couvrant de G de poids minimum. Préciser l'ordre dans lequel des arêtes sont considérées et éventuellement sélectionnées.

arête	ab	ad	bc	bd	be	ce	de	df	dg	eg	eh	fg	gh
poids	2	4	1	3	3	2	4	4	3	3	4	1	2

3 Détection de cycles pairs

Il est connu que l'on peut utiliser un parcours en largeur pour détecter la présence (ou l'absence) de cycles impairs dans un graphe.

3.1) Proposer une modification d'un des deux algorithmes d'exploration de graphes (bien réfléchir avant de choisir), qui détecte si un graphe donné contient un cycle de longueur paire. Expliciter l'algorithme soit en ajoutant/modifiant des lignes d'un algorithme existant, soit en proposant un nouvel algorithme faisant appel à un algorithme existant.

3.2) Étudier la complexité de votre algorithme.

4 Optimisation de flot

La matrice C ci-dessous à gauche est la matrice d'adjacence d'un réseau de flot G : le premier sommet est la source, le dernier le puits ; une valeur strictement positive représente la capacité de l'arc correspondant. La matrice F ci-dessous à droite décrit un flot f sur G.

$$C = \begin{bmatrix} 0 & 4 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 3 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad F = \begin{bmatrix} 0 & 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4.1) Dessiner le graphe G en représentant pour chaque arc le flot actuel et la capacité.

4.2) Le flot actuel est-il optimal ? Justifier. Si ce n'est pas le cas, appliquer l'algorithme de Ford-Fulkerson jusqu'à ce que le flot soit optimal. Une fois le flot optimal, expliciter une coupe minimale justifiant l'optimalité du flot.

5 Chemins arc-disjoints

Soit G un graphe orienté avec une source s (un sommet sans arc entrant) et un puits t (un sommet sans arc sortant). Deux chemins reliant s à t sont dits arc-disjoints, s'ils ne partagent pas d'arc (cependant, ils peuvent passer tous les deux par un même sommet).

On souhaite connaître le plus grand nombre possible de chemins élémentaires arc-disjoints reliant s à t. Soit (G, s, t, c) le réseau de flot corresponant au graphe donné G, où la capacité c de tout arc du graphe est égale à 1.

5.1) Montrer que s'il existe k chemins arc-disjoints reliant s à t dans un graphe G, alors il existe un flot de valeur k dans le réseau de flot G.

Le reste de l'exercice a pour objectif de démontrer la réciproque : s'il existe un flot de valeur k dans G, alors il existe k chemins arc-disjoints reliant s à t.

Un flot est dit *binaire* si le flot $f(e)$ sur chaque arc d'un réseau est soit 0 soit 1. Si f est un flot binaire, on notera $E_1(f)$ l'ensemble des arcs avec un flot de 1.

5.2) Soit f un flot binaire de valeur $k \geq 1$. Montrer que l'ensemble $E_1(f)$ contient un chemin élémentaire de s à t.

5.3) Soit f un flot binaire de valeur $k \geq 1$. Montrer que si l'ensemble $E_1(f)$ contient un circuit C, alors en annulant le flot le long de C le résultat obtenu reste un flot.

5.4) En déduire que s'il existe un flot binaire de valeur $k \geq 1$, alors il existe un flot f' de même valeur que f tel que $E_1(f')$ ne contient aucun circuit.

5.5) Déduire des questions précédentes que s'il existe un flot binaire de valeur k, alors il existe k chemins arc-disjoints reliant s et t.

5.6) Montrer que si toutes les capacités d'arc sont binaires (soit 0 soit 1), alors une étape d'amélioration dans l'algorithme de Ford-Fulkerson transforme un flot binaire en un flot binaire. En déduire que le flot retourné par l'algorithme de Ford-Fulkerson est binaire.

5.7) En déduire le théorème de Menger : Soit G un graphe orienté et soient s et t deux sommets de G. Le nombre maximum de chemins élémentaires arc-disjoints reliant s à t est égal au nombre minimum d'arcs dont la suppression rend t inaccessible depuis s.

Algorithme 1 Parcours en largeur PL(G, s)

```
1: couleur(s) ← GRIS
2: d(s) ← 0
3: pere(s) ← NIL
4: F ← {s}
5: pour tout v ∈ V(G)\s faire
6:    couleur(v) ← BLANC
7:    d(v) ← ∞
8:    pere(v) ← NIL
9: fin pour
10: tant que F non vide faire
11:    v ← tete(F)
12:    pour tout w ∈ Adj(v) faire
13:       si couleur(w) = BLANC alors
14:          couleur(w) ← GRIS
15:          d(w) ← d(v) + 1
16:          pere(w) ← v
17:          Enfiler(F, w)
18:       fin si
19:    fin pour
20:    Defiler(F)
21:    couleur(v) ← NOIR
22: fin tant que
```

Algorithme 2 Parcours en profondeur PP(G)

pour tout $v \in V(G)$ **faire**
  $couleur(v) \leftarrow BLANC$
  $pere(v) \leftarrow NIL$
fin pour
$temps \leftarrow 0$
pour tout $v \in V(G)$ **faire**
  si $couleur(v) = BLANC$ **alors**
    $VisiterPP(v)$
  fin si
fin pour

Algorithme 3 VisiterPP(v)

$d(v) \leftarrow temps \leftarrow temps + 1$
$couleur(v) \leftarrow GRIS$
pour tout $w \in Adj(v)$ **faire**
  si $couleur(w) = BLANC$ **alors**
    $pere(w) \leftarrow v$
    $VisiterPP(w)$
  fin si
fin pour
$couleur(v) \leftarrow NOIR$
$f(v) \leftarrow temps \leftarrow temps + 1$

Algorithme 4 CFC(G)

Exécuter PP(G) et trier les sommets selon un ordre décroissant de f
Calculer G^{-1}
Exécuter PP(G^{-1})
Retourner les arborescences obtenues

Algorithme 5 Kruskal(G, w)

```
1: trier les arêtes de G par poids croissant dans une file L
2: pour tout u ∈ V(G) faire
3:    cc(u) ← u
4: fin pour
5: tant que L non vide faire
6:    soit (u, v) la tête de L
7:    si cc(u) ≠ cc(v) alors
8:       ajouter l'arête (u, v) à l'arbre
9:       pour tout x ∈ V(G) faire
10:         si cc(x) = cc(v) alors
11:            cc(x) ← cc(u)
12:         fin si
13:      fin pour
14:   fin si
15:   Défiler (u, v)
16: fin tant que
```

Algorithme 6 Prim(G, w, r)

```
1: pour tout sommet v faire
2:    dist_T(v) ← ∞
3:    si v voisin de s alors
4:       dist_T(v) ← w(r, v)
5:       cible(v) ← r
6:    fin si
7: fin pour
8: dist_T(r) ← 0
9: pour tout i de 1 à n − 1 faire
10:   sélectionner le sommet v tel que dist_T(v) > 0 minimum
11:   ajouter l'arête (v, cible(v)) à l'arbre
12:   dist_T(v) ← 0
13:   pour tout voisin x de v faire
14:      si dist_T(x) > w(v, x) alors
15:         dist_T(x) ← w(v, x)
16:         cible(x) ← v
17:      fin si
18:   fin pour
19: fin pour
```

Algorithme 7 Ford-Fulkerson(G, s, t, c)

```
1: Marguage()
2: tant que le sommet destination t est marqué faire
3:    Améliorer()
4:    Marquage()
5: fin tant que
```

Algorithme 8 Marquage()

```
1: pour chaque sommet v faire
2:    démarquer(v)
3: fin pour
4: marquer(s)
5: enfiler(s)
6: tant que file non vide faire
7:    u ← tête de la file
8:    pour chaque voisin sortant v de u non-marqué faire
9:       si f(uv) < c(uv) alors
10:         marquer(v)
11:         enfiler(v)
12:         π(v) ← uv
13:      fin si
14:   fin pour
15:   pour chaque voisin entrant v de u non-marqué faire
16:      si f(vu) > 0 alors
17:         marquer(v)
18:         enfiler(v)
19:         π(v) ← vu
20:      fin si
21:   fin pour
22:   défiler(u)
23: fin tant que
```

Algorithme 9 Améliorer()

```
1: v ← t
2: val ← ∞
3: tant que v ≠ s faire
4:    si π(v) = uv alors
5:       val ← min{val, c(uv) − f(uv)}
6:    fin si
7:    si π(v) = vu alors
8:       val ← min{val, f(uv)}
9:    fin si
10:   v ← u
11: fin tant que
12: v ← t
13: tant que v ≠ s faire
14:   si π(v) = uv alors
15:      f(uv) ← f(uv) + val
16:   fin si
17:   si π(v) = vu alors
18:      f(vu) ← f(vu) − val
19:   fin si
20:   v ← u
21: fin tant que
```