université de BORDEAUX

Année 2023-2024 — Session de décembre 2023
Étape : INF501-MI501 — UE : 4TIN501U
Épreuve de : Algorithmique des graphes
Date : 20 décembre 2023 – 9h — **Durée :** 1h30
Épreuve de : F. Kardoš — **Nb. pages : 7**

Aucun document autorisé.
Une attention toute particulière sera portée à la présentation et à la rédaction de la copie.
Les algorithmes sont rappelés à la fin du document.

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

1 Arbre couvrant de poids minimum

Soit G_1 le graphe (non-orienté) dont les arêtes avec leur poids sont représentées dans le tableau ci-dessous.

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

1.1) Dessiner G_1. Donner le degré minimum et le degré maximum de G_1.

1.2) Est-il connexe ? Combien d'arêtes doit contenir un arbre couvrant de G_1 ?

1.3) Trouver un arbre couvrant de G_1 de poids minimum par un algorithme de votre choix (soit Kruskal soit Prim, les deux étant rappelés en annexe). Expliciter l'ordre dans lequel les arêtes conservées ont été sélectionnées.

1.4) Quel est le poids total d'un tel arbre ? La solution est-elle unique ?

2 Composantes fortement connexes

2.1) Rappeler la définition de la matrice d'adjacence d'un graphe orienté.

Soit G_2 le graphe orienté représenté par la matrice ci-contre.

2.2) Quel est le degré entrant et le degré sortant du sommet s_0 ?

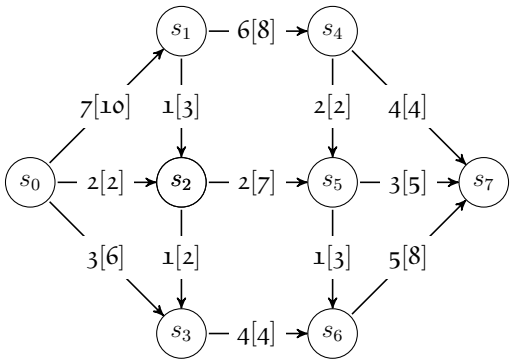
	s_0	s_1	s_2	s_3	s_4	s_5	s_6
s_0	0	1	0	0	0	0	0
s_1	0	0	1	0	0	0	0
s_2	1	1	0	1	0	0	0
s_3	0	1	0	0	0	0	1
s_4	1	0	0	0	0	1	0
s_5	0	0	0	1	1	0	1
s_6	0	0	0	0	0	0	1

2.3) Appliquer l'algorithme vu en cours (rappelé en annexe) pour déterminer les composantes fortement connexes du graphe G_2.
Expliciter l'ordre dans lequel les sommets sont considérés lors du deuxième parcours en profondeur.

2.4) Dessiner G_2 en explicitant ses composantes fortement connexes en les encerclant.

3 Optimisation de flot

Les arcs du graphe G_3 sont valués par des nombres entre crochets (représentant leur capacité), et des nombres représentant un flot compatible avec les capacités.



3.1) Utiliser l'algorithme de *Ford-Fulkerson* pour augmenter le flot indiqué de façon à obtenir un flot maximum allant du sommet s_0 au sommet s_7 (vous indiquerez les augmentations successives du flot, et vous mettrez en évidence le fait que le dernier flot obtenu est maximum en dessinant par exemple une coupe minimale).

Nous allons maintenant s'intéresser à l'évolution de la valeur d'un flot optimal en fonction de changement de capacité de l'arc s_2s_5.

3.2) Si la capacité de l'arc s_2s_5 augmente, est-ce que la valeur d'un flot optimal augmente aussi ? Justifier.

3.3) Pour quelles valeurs c comme nouvelle capacité de l'arc s_2s_5 la valeur d'un flot optimal ne change pas même si la capacité est plus petite que la capacité actuelle ?

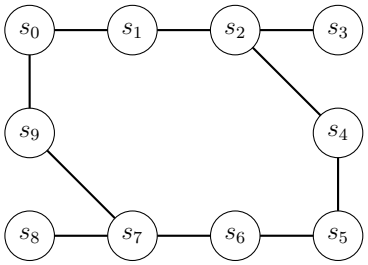
3.4) Pour quelles valeurs c comme nouvelle capacité de l'arc s_2s_5 la valeur d'un flot optimal diminue forcément ? Quel est cette nouvelle valeur d'un flot optimal en fonction de c ?

4 Excentricité et diamètre

Soit G un graphe non-orienté connexe. L'*excentricité* d'un sommet u est définie comme la plus grande distance entre u et un autre sommet de G :

$$ex(u) = \max_{v \in V(G)} dist(u, v).$$

Par exemple, dans le graphe G_4 ci-contre, l'excentricité du sommet s_0 est de 4, alors que celle du sommet s_2 est de 5.



Le diamètre d'un graphe G connexe, $diam(G)$, est défini comme la plus grande excentricité de ses sommets :

$$diam(G) = \max_{u \in V(G)} ex(u) = \max_{u,v \in V(G)} dist(u, v).$$

4.1) Donner le diamètre du graphe G_4.

4.2) Proposer un algorithme, en se basant sur l'algorithme de parcours en largeur, qui, étant donné un graphe G supposé connexe et un sommet s, calcule et retourne l'excentricité du sommet s.

4.3) Proposer un algorithme qui, en faisant appel à l'algorithme précédent, calcule et retourne le diamètre d'un graphe donné, supposé connexe.

4.4) Étudier la complexité de votre algorithme.

Dans la suite, nous allons mettre en place un algorithme plus efficace pour calculer le diamètre d'un arbre. Il est basé sur la proposition suivante :

Théorème 1 *Soit G un arbre de diamètre d, soit u un sommet de G. Soit v un sommet de G tel que $dist(u, v) = ex(u)$. Alors $ex(v) = d$.*

4.5) Admettons que Théorème 1 soit vrai. Proposer un algorithme qui, étant donné un arbre G, calcule le diamètre de G en faisant seulement deux parcours en largeur.

4.6) Quelle est la complexité de ce nouvel algorithme ?

4.7) Démontrer le théorème 1. (Vous pouvez essayer par l'absurde : En supposant que pour tout sommet v à distance $ex(u)$ de u on a $ex(v) < d$, considérer deux autres sommets x et y à distance $diam(G)$ entre eux ; étudier les distances entre u, v, x, et y pour, en s'appuyant sur le fait que G soit un arbre, trouver une contradiction.)

Algorithmes

Algorithme 1 Kruskal(G, w)

```
1:  pour tout v de V(G) faire
2:      composante(v) ← {v}
3:  fin pour
4:  ncomposantes ← n
5:  Trier E(G) dans un ordre croissant (e_1, ..., e_m) en fonction de w(e)
6:  i ← 1
7:  E(T) ← {}
8:  tant que ncomposantes > 1 faire
9:      si e_i = (u, v) et composante(u) ≠ composante(v) alors
10:         E(T) ← E(T) ∪ {e_i}
11:         UNIFIER(composante(u), composante(v))
12:         ncomposantes ← ncomposantes − 1
13:     fin si
14:     i ← i + 1
15: fin tant que
16: retourner E(T)
```

Algorithme 2 Prim(G, w, r)

```
1:  pour tout sommet v faire
2:      dist_T(v) ← ∞
3:      si v voisin de r 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 3 Parcours en profondeur PP(G)

1: **pour tout** $v \in V(G)$ **faire**
2: $couleur(v) \leftarrow BLANC$
3: $pere(v) \leftarrow NIL$ f
4: **fin pour**
5: $temps \leftarrow 0$
6: **pour tout** $v \in V(G)$ **faire**
7: **si** $couleur(v) = BLANC$ **alors**
8: $VisiterPP(v)$
9: **fin si**
10: **fin pour**

Algorithme 4 VisiterPP(v)

1: $d(v) \leftarrow temps \leftarrow temps + 1$
2: $couleur(v) \leftarrow GRIS$
3: **pour tout** $w \in Adj(v)$ **faire**
4: **si** $couleur(w) = BLANC$ **alors**
5: $pere(w) \leftarrow v$
6: $VisiterPP(w)$
7: **fin si**
8: **fin pour**
9: $couleur(v) \leftarrow NOIR$
10: $f(v) \leftarrow temps \leftarrow temps + 1$

Algorithme 5 Composantes Fortement Connexes(G)

1: PP(G)
2: Calculer G^{-1}
3: PP(G^{-1}) en considérant les sommets dans l'ordre décroissant des $f(u)$ obtenus par PP(G) à l'étape 1
4: Afficher les arborescences de la forêt obtenue à l'étape 3

Dans l'Algorithme 6 (FlotMax), le paramètre c désigne les capacités du graphe G, s la source et t la destination du flot f calculé. $I(e)$ et $T(e)$ désignent respectivement le sommet initial et le sommet terminal d'un arc e.

Algorithme 6 FlotMax(G, c, s, t)

```
1: pour tout e de E(G) faire
2:    f[e] ← 0
3: fin pour
4: répéter
5:    Marquage(G, c, f, s, t)
6:    si t ∈ Y alors
7:       v ← t
8:       C⁺ ← {(t, s)}
9:       C⁻ ← ∅
10:      tant que v ≠ s faire
11:         e ← A[v]
12:         si v = T[e] alors
13:            C⁺ ← C⁺ ∪ {e}
14:            v ← I[e]
15:         sinon
16:            C⁻ ← C⁻ ∪ {e}
17:            v ← T[e]
18:         fin si
19:      fin tant que
20:   fin si
21:   pour tout e ∈ C⁺ faire
22:      f(e) ← f(e) + δ[t]
23:   fin pour
24:   pour tout e ∈ C⁻ faire
25:      f(e) ← f(e) − δ[t]
26:   fin pour
27: jusqu'à t ∉ Y
```

Algorithme 7 Marquage(G, c, f, s, t)

$Y \leftarrow \{s\}$
$\delta(s) \leftarrow +\infty$
$Max \leftarrow$ faux
tant que $t \notin Y et Max =$faux **faire**
  si il existe $e = (u, v)$ avec $u \in Y, v \notin Y, f(e) < c(e)$ **alors**
    $Y \leftarrow Y \cup \{v\}$
    $A[v] \leftarrow e$
    $\delta[v] \leftarrow min(\delta[u], c(e) - f(e))$
  sinon
    si il existe $e = (u, v)$ avec $v \in Y, u \notin Y, f(e) > 0$ **alors**
      $Y \leftarrow Y \cup \{u\}$
      $A[u] \leftarrow e$
      $\delta[u] \leftarrow min(\delta[v], f(e))$
    sinon
      Max $\leftarrow vrai$
    fin si
  fin si
fin tant que

Algorithme 8 Parcours en largeur PL(G, s)

$couleur(s) \leftarrow GRIS$
$d(s) \leftarrow 0$
$pere(s) \leftarrow NIL$
$F \leftarrow \{s\}$
pour tout $v \in V(G) \backslash s$ **faire**
  $couleur(v) \leftarrow BLANC$
  $d(v) \leftarrow \infty$
  $pere(v) \leftarrow NIL$
fin pour
tant que F non vide **faire**
  $v \leftarrow tete(F)$
  pour tout $w \in Adj(v)$ **faire**
    si $couleur(w) = BLANC$ **alors**
      $couleur(w) \leftarrow GRIS$
      $d(w) \leftarrow d(v) + 1$
      $pere(w) \leftarrow v$
      $Enfiler(F, w)$
    fin si
  fin pour
  $Defiler(F)$
  $couleur(v) \leftarrow NOIR$
fin tant que