université de BORDEAUX

Année 2019-2020 — Session de décembre 2019
Étape : INF501-MI501 — UE : 4TIN501U
Épreuve de : Algorithmique des graphes
Date : 19 décembre 2019 – 9h00 — **Durée :** 1h30
Épreuve de : F. Kardoš — **Nb. pages : 8**

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 (resp. des arcs) d'un graphe non orienté (resp. orienté) G. Les variables n et m désigneront respectivement le nombre de sommets et d'arêtes (resp. d'arcs). Par défaut, dans toutes les structures de données les sommets sont stockés et parcourus dans un ordre alphabétique.

1 Distances avec Dijkstra

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 orienté.

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

1.2) Dans le tableau ci-dessous se trouvent les poids des arcs de G. Calculer les distances de tous les sommets à partir du sommet a. Préciser l'ordre dans lequel des sommets sont visités par l'algorithme.

arc	ab	ad	bc	bd	be	ce	de	df	dg	eg	eh	fg	gh
poids	2	5	1	2	4	2	4	2	3	3	4	1	2

2 Optimisation de flot

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 orienté.

2.3) Dessiner G. Contient-il un circuit ?

2.4) Dans le tableau ci-dessous se trouvent les capacités des arcs de G. Calculer le flot optimal de a à h. Préciser les chemins améliorants, ainsi qu'une coupe (minimale) justifiant que le flot obtenu est maximum.

arc	ab	ad	bc	bd	be	ce	de	df	dg	eg	eh	fg	gh
capacité	2	5	1	2	4	2	4	2	3	3	4	1	2

3 Distances avec Floyd et reverse engineering

Voici le contenu des matrices W (les distances) et Π (les prédécesseurs) à l'issue d'un appel à l'algorithme de Floyd sur un graphe orienté G à 4 sommets.

W	A	B	C	D
A	0	1	1	-1
B	1	0	0	-2
C	4	3	0	1
D	3	4	2	0

Π	A	B	C	D
A	A	A	D	B
B	D	B	D	B
C	D	C	C	B
D	D	A	D	D

3.1) Expliciter le chemin optimal de A vers C et pour chaque arc qui le compose calculer son poids.

3.2) Expliciter le chemin optimal de C vers A et pour chaque arc qui le compose calculer son poids.

3.3) Dessiner le graphe G, sachant qu'il y a cinq arcs au total.

4 Détection d'isthmes

Soit G un graphe non orienté. Une arête $e = uv$ est un *isthme* si la suppression de l'arête e augmente le nombre de composantes connexes de G.

4.4) Montrer que e est un isthme si et seulement si e n'est contenue dans aucun cycle de G.

Étant donné un graphe G et une arête $e = uv$ de G, considérons maintenant la procédure d'étiquetage des sommets de G suivante : Initialement, le sommet u est étiqueté 0, le sommet v est étiqueté 1, et les autres sommets n'ont pas d'étiquette. Ensuite, tant qu'il existe un sommet sans étiquette, disons x, ayant un voisin étiqueté, soit y, le sommet x est étiqueté avec l'étiquette du sommet y. (Si un sommet x a plusieurs voisins portant déjà des étiquettes, il peut prendre n'importe quelle étiquette parmi celles-là).

4.5) Montrer qu'à l'issue de la procédure, un sommet est étiqueté si et seulement s'il appartient à la composante connexe de G contenant e.

4.6) Montrer qu'une arête e est un isthme si et seulement si à l'issue de la procédure, e est la seule arête reliant deux sommets portant deux étiquettes différentes.

4.7) Proposer un algorithme qui, étant donné un graphe G et une arête e, en temps $O(n + m)$ décide si e est un isthme ou pas.

5 Modification de l'algorithme de Kruskal

Soit G un graphe ayant des arêtes munies d'une fonction de poids w.

On rappelle qu'un graphe partiel de G est un graphe contenant tous les sommets de G et seulement une partie des arêtes de G.

5.1) Si G contient des arêtes de poids négatifs, le graphe partiel connexe de poids minimum de G est-il toujours un arbre ? Justifier votre réponse.

5.2) Modifier la version de l'algorithme de Kruskal donnée ci-dessous pour l'adapter à un graphe possédant des arêtes de poids négatifs.

5.3) Appliquer votre algorithme au graphe ci-dessous. On donnera l'ordre des arêtes rajoutés au graphe partiel connexe obtenu.

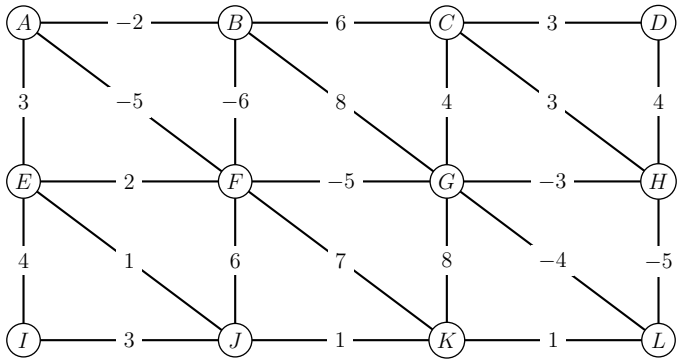

FIGURE 1 – Graphe G

```
0   Kruskal(G,w,s)
1     pour chaque sommet v de V(G) faire
2         composante(v) <- {v}
3     fin pour
4     trier E(G) dans un ordre croissant (e1, ..., em) en fonction de w(e)
5     i <- 1
6     E(H) <- {}
7     ncomposantes <- n
8     tant que ncomposantes > 1 faire
9         si ei = (u, v) et composante(u) != composante(v) alors
10            E(H) <- E(H) U {ei}
11            Unifier(composante(u), composante(v))
12            ncomposantes <- ncomposantes - 1
13        fin si
14    fin tant que
15    retourner E(T)
```

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)

```
1: pour tout v ∈ V(G) faire
2:    couleur(v) ← BLANC
3:    pere(v) ← NIL
4: fin pour
5: temps ← 0
6: pour tout v ∈ V(G) faire
7:    si couleur(v) = BLANC alors
8:       VisiterPP(v)
9:    fin si
10: fin pour
```

Algorithme 3 VisiterPP(v)

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

Algorithme 4 Dijkstra(G, w, s)

```
1: pour tout v de V(G) faire
2:    d(v) ← ∞
3:    pere(v) ← NIL
4:    couleur(v) ← BLANC
5: fin pour
6: d(s) ← 0
7: F ← FILE_PRIORITE({s}, d)
8: tant que F ≠ ∅ faire
9:    pivot ← EXTRAIRE_MIN(F)
10:   pour tout e = (pivot, v) arc sortant de pivot faire
11:      si couleur(v) = BLANC alors
12:         si d(v) = ∞ alors
13:            INSERER(F, v)
14:         fin si
15:         si d[v] > d[pivot] + w(e) alors
16:            d[v] ← d[pivot] + w(e)
17:            pere[v] ← pivot
18:         fin si
19:      fin si
20:   fin pour
21:   couleur[pivot] ← NOIR
22: fin tant que
```

Algorithme 5 Floyd(G, w)

```
1: pour i de 1 à n faire
2:    pour j de 1 à n faire
3:       si i = j alors
4:          W(i,j) ← 0
5:          Π(i,j) ← i
6:       sinon si ij ∈ E(G) alors
7:          W(i,j) ← w(i,j)
8:          Π(i,j) ← i
9:       sinon
10:         W(i,j) ← ∞
11:         Π(i,j) ← NIL
12:      fin si
13:   fin pour
14: fin pour
15: pour k de 1 à n faire
16:   pour i de 1 à n faire
17:      pour j de 1 à n faire
18:         si W(i,k) + W(k,j) < W(i,j) alors
19:            W(i,j) ← W(i,k) + W(k,j)
20:            Π(i,j) ← Π(k,j)
21:         fin si
22:      fin pour
23:   fin pour
24: fin pour
```

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)

```
1: Y ← {s}
2: δ(s) ← +∞
3: Max ← faux
4: tant que t ∉ Y et Max =faux faire
5:    si il existe e = (u, v) avec u ∈ Y, v ∉ Y, f(e) < c(e) alors
6:       Y ← Y ∪ {v}
7:       A[v] ← e
8:       δ[v] ← min(δ[u], c(e) − f(e))
9:    sinon
10:      si il existe e = (u, v) avec v ∈ Y, u ∉ Y, f(e) > 0 alors
11:         Y ← Y ∪ {u}
12:         A[u] ← e
13:         δ[u] ← min(δ[v], f(e))
14:      sinon
15:         Max ←vrai
16:      fin si
17:   fin si
18: fin tant que
```