université de BORDEAUX

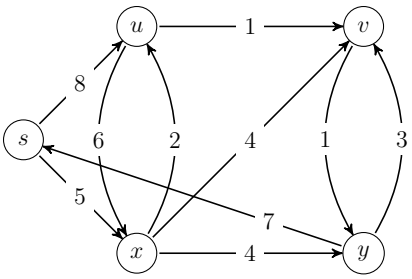
Année 2022-2023 — Session de décembre 2022
Étape : INF501-MI501 — UE : 4TIN501U
Épreuve de : Algorithmique des graphes
Date : 13 décembre 2022 – 9h — **Durée :** 1h30
Épreuve de : F. Kardoš — **Nb. pages : 8**

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

Soit $G_1 = (X_1, A_1)$ le graphe orienté avec une fonction de pondération $w : A_1 \to \mathbb{R}$ représenté ci-dessous.



1.1) En appliquant l'algorithme de *Dijkstra*, déterminer la distance (longueur du plus court chemin) du sommet s à chacun des sommets de G_1.
Spécifier l'ordre dans lequel les sommets du graphe sont traités par l'algorithme.
1.2) Dessiner l'arborescence des plus courts chemins obtenus.

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.

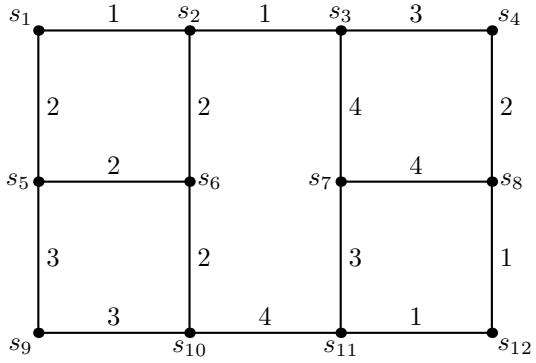
2.1) Dessiner G. Contient-il un circuit ?

2.2) 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	7	5	1	2	4	3	2	1	3	5	7	2	6

3 Arbre de poids minimum

Soit G_3 le graphe non-orienté valué dessiné ci-dessous.



3.1) Trouver un arbre couvrant de G_3 de poids minimum par un algorithme de votre choix (soit Kruskal soit Prim, les deux rappelés en annexe). Expliciter l'ordre dans lequel les arêtes conservées ont été sélectionnées. Quel est le poids total d'un tel arbre ? La solution est-elle unique ?

3.2) Montrer que, d'une manière générale, en ajoutant une nouvelle arête (d'un poids quelconque) à un graphe G donné, si le poids total d'un arbre couvrant de poids minimum est modifié, il ne peut que diminuer.

Ajoutons maintenant dans le graphe de Question 3.1 une nouvelle arête reliant les sommets s_6 et s_7 de poids $b \geq 0$.

3.3) Pour quelles valeurs de b (à préciser sous la forme d'un intervalle) cette nouvelle arête est-elle comprise dans **tout** arbre de poids minimum ? Quel est le poids total d'un tel arbre (en fonction de b) ?

3.4) Pour quelles valeurs de b (à préciser sous la forme d'un intervalle) cette nouvelle arête n'est-elle comprise dans **aucun** arbre de poids minimum ? Quel est le poids total d'un tel arbre (en fonction de b) ?

3.5) Existe-t-il une valeur de b pour laquelle la nouvelle arête est comprise dans un arbre de poids minimum, bien qu'il existe un autre arbre de poids minimum qui évite cette nouvelle arête ? Si c'est le cas, donner de tels arbres ; sinon, justifier votre réponse.

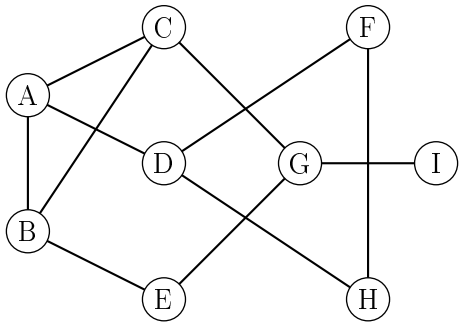
4 Détection d'isthmes

Tous les graphes dans cet exercice sont non-orientés.

L'objectif de cet exercice est de proposer un algorithme pour identifier les isthmes dans un graphe donné.

4.1) Rappeler la définition d'un graphe *connexe*, et d'une *composante connexe* d'un graphe.

Un *isthme* est une arête dont la suppression augmente le nombre de composantes connexes. On rappelle que, par exemple, dans un arbre, toute arête est un isthme.



4.2) Identifier tous les isthmes du graphe G_4 dessiné ci-dessus.

On rappelle qu'un *point d'articulation* est un sommet dont la suppresion (entraînant la suppression de toutes les arêtes incidentes) augmente le nombre de composantes connexes.

4.3) Montrer qu'une arête e est un isthme si et seulement si elle relie deux sommets distincts dont chacun est soit un point d'articulation soit une feuille (c-à-d un sommet de degré 1). *Attention, il faut démontrer deux implications.*

On rappelle que lors d'un parcours en profondeur d'un graphe non-orienté, toute arête d'un graphe donné devient soit une arête de liaison (une arête de découverte) reliant un sommet à un de ses fils, soit une arête de retour reliant un sommet à un de ses ancêtres (y compris son père dans le cas d'arêtes parallèles, voire lui-même dans le cas d'une boucle).

4.4) Montrer qu'un isthme est toujours une arête de liaison dans tout parcours en profondeur.

Un exercice de TD avait pour sujet de montrer qu'il est possible d'identifier les points d'articulation d'un graphe non-orienté donné par un algorithme de complexité $O(n+m)$, basé sur un parcours en profondeur.

Plus précisément, il est possible de décider pour chaque sommet d'un graphe donné s'il est un point d'articulation au moment de la fin de sa visite, à condition de calculer, au fur et à mesure, pour tout sommet, la date de début de visite d'un ancêtre le plus ancien vers lequel il est possible de remonter par une arête de retour.

Une implémentation d'un tel algorithme est rappelé en annexe (voir Algorithme 6 et 7).

4.5) Proposer une modification de l'Algorithme 6 et 7 pour qu'il identifie tous les isthmes du graphe ; votre algorithme doit rester de complexité $O(n+m)$. Expliciter les lignes de code à ajouter/modifier. Justifier que la complexité ne change pas en étudiant la compexité des morceaux de code ajoutés.

Algorithmes

Algorithme 1 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
```

Dans l'Algorithme 2 (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 2 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 3 Marquage(G, c, f, s, t)

1: $Y \leftarrow \{s\}$
2: $\delta(s) \leftarrow +\infty$
3: $Max \leftarrow$ faux
4: **tant que** $t \notin Y et Max =$faux **faire**
5: **si** il existe $e = (u, v)$ avec $u \in Y, v \notin Y, f(e) < c(e)$ **alors**
6: $Y \leftarrow Y \cup \{v\}$
7: $A[v] \leftarrow e$
8: $\delta[v] \leftarrow min(\delta[u], c(e) - f(e))$
9: **sinon**
10: **si** il existe $e = (u, v)$ avec $v \in Y, u \notin Y, f(e) > 0$ **alors**
11: $Y \leftarrow Y \cup \{u\}$
12: $A[u] \leftarrow e$
13: $\delta[u] \leftarrow min(\delta[v], f(e))$
14: **sinon**
15: Max $\leftarrow vrai$
16: **fin si**
17: **fin si**
18: **fin tant que**

Algorithme 4 Kruskal(G, w)

pour tout v de $V(G)$ **faire**
  $composante(v) \leftarrow \{v\}$
fin pour
$ncomposantes \leftarrow n$
Trier $E(G)$ dans un ordre croissant $(e_1, \ldots, e_m)$ en fonction de $w(e)$
$i \leftarrow 1$
$E(T) \leftarrow \{\}$
tant que $ncomposantes > 1$ **faire**
  si $e_i = (u, v)$ et $composante(u) \neq composante(v)$ **alors**
    $E(T) \leftarrow E(T) \cup \{e_i\}$
    UNIFIER$(composante(u), composante(v))$
    $ncomposantes \leftarrow ncomposantes - 1$
  fin si
  $i \leftarrow i + 1$
fin tant que
retourner $E(T)$

Algorithme 5 Prim(G, w, r)

pour tout sommet v **faire**
  $dist_T(v) \leftarrow \infty$
  si v voisin de r **alors**
    $dist_T(v) \leftarrow w(r, v)$
    $cible(v) \leftarrow r$
  fin si
fin pour
$dist_T(r) \leftarrow 0$
pour tout i de 1 à $n - 1$ **faire**
  sélectionner le sommet v tel que $dist_T(v) > 0$ minimum
  ajouter l'arête $(v, cible(v))$ à l'arbre
  $dist_T(v) \leftarrow 0$
  pour tout voisin x de v **faire**
    si $dist_T(x) > w(v, x)$ **alors**
      $dist_T(x) \leftarrow w(v, x)$
      $cible(x) \leftarrow v$
    fin si
  fin pour
fin pour

Algorithme 6 Points d'articulation(G)

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

Algorithme 7 VisiterPP(v)

```
1: ℓ(v) ← 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:       si pere(v) = NIL et temps > d[v] alors
7:          PA(v) ← Vrai
8:       fin si
9:       VisiterPP(w)
10:      si ℓ(w) < ℓ(v) alors
11:         ℓ(v) ← ℓ(w)
12:      fin si
13:      si ℓ(w) ≥ d(v) et pere(v) ≠ NIL alors
14:         PA(v) ← Vrai
15:      fin si
16:   sinon
17:      si couleur(w) = GRIS et w ≠ pere(v) et d(w) < ℓ(v) alors
18:         ℓ(v) ← d(w)
19:      fin si
20:   fin si
21: fin pour
22: couleur(v) ← NOIR
23: f(v) ← temps ← temps + 1
```