

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 d'un graphe G . Les variables n et m désigneront respectivement le nombre de sommets et d'arêtes.

1 Composantes fortement connexes

1.1) En utilisant l'algorithme de calcul des composantes fortement connexes (Algorithme 1) basé sur le parcours en profondeur (Algorithmes 2 et 3), donnez les composantes fortement connexes du graphe de la Figure 1. Pour chaque parcours en profondeur, on dessinera l'arborescence et donnera les heures de début et fin de visite. On supposera que les listes des successeurs sont données dans l'ordre lexicographique et que le premier sommet visité lors du premier parcours en profondeur est le sommet A . On rappelle que le graphe G^{-1} désigne le graphe inverse de G , c'est à dire le graphe $(V(G), E(G^{-1}))$, avec $\forall u, v \in V(G), (u, v) \in E(G^{-1}) \iff (v, u) \in E(G)$.

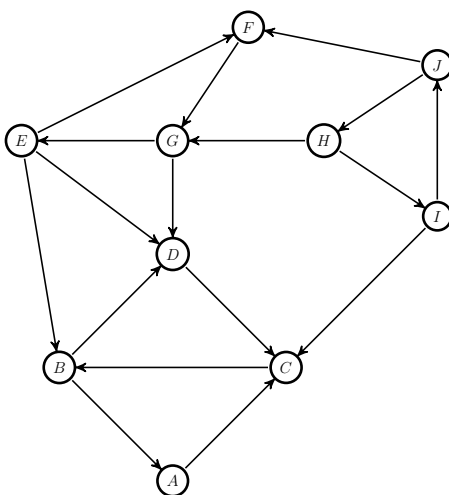


FIGURE 1 – Graphe G

Algorithme 1 Composantes fortement connexes CFC(G)

- 1: Exécuter PP(G)
 - 2: Calculer G^{-1}
 - 3: Exécuter PP(G^{-1}) en considérant les sommets dans la boucle 6 à 10 de PP(G^{-1}) dans l'ordre décroissant de la fonction f (fin de visite) obtenue à l'étape 1.
 - 4: Retourner les arborescences obtenues comme composantes fortement connexes de G
-

Algorithme 2 Parcours en profondeur PP(G)

- 1: **pour tout** $v \in V(G)$ **faire**
 - 2: $couleur(v) \leftarrow BLANC$
 - 3: $pere(v) \leftarrow NIL$
 - 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 3 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: $f(v) \leftarrow temps \leftarrow temps + 1$
-

2 Algorithme de Prim

2.1) En utilisant l'algorithme de Prim (Algorithme 4), trouver l'arbre de poids minimum dans le graphe de la Figure 2, en partant du sommet A . Donner pour chaque étape le nom du sommet extrait de la file de priorité et les changements de valeurs pour la fonction *clé*. Dessiner l'arbre de poids minimum obtenu.

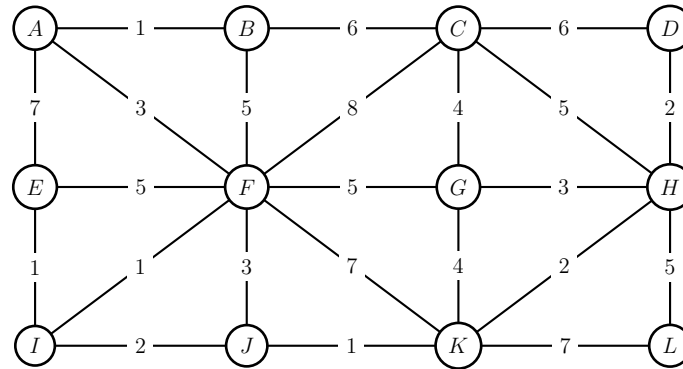


FIGURE 2 – Graphe G

Algorithme 4 Prim(G, w)

```

1:  $F \leftarrow \text{FILE\_PRIORITE}(V(G), \text{clé})$ 
2: pour tout  $v$  de  $V(G)$  faire
3:    $\text{clé}(v) \leftarrow \infty$ 
4: fin pour
5:  $\text{clé}(r) \leftarrow 0$ 
6:  $\text{père}(r) \leftarrow \text{NIL}$ 
7: tant que  $F \neq \emptyset$  faire
8:    $u \leftarrow \text{EXTRAIRE\_MIN}(F)$ 
9:   pour tout  $v \in \text{Adj}(u)$  faire
10:    si  $v \in F$  et  $w(u, v) < \text{clé}(v)$  alors
11:       $\text{père}(v) \leftarrow u$ 
12:       $\text{clé}(v) \leftarrow w(u, v)$ 
13:    fin si
14:   fin pour
15: fin tant que

```

3 Flot Maximum

3.1) En utilisant l'algorithme de Ford-Fulkerson (Algorithmes 5 et 6), trouver le flot maximum entre les sommets s et t du réseau de la Figure 3. Le flot possède déjà une valeur initiale donnée par le premier nombre porté sur les arcs, le second étant la capacité de l'arc. Par exemple, l'arc (s, A) possède un flot initial de 1 et une capacité de 4. *Vous devez améliorer ce flot existant, sans repartir d'un flot nul.* On donnera pour chaque exécution de la procédure de marquage le chemin augmentant obtenu et l'augmentation correspondante.

3.2) Donner la coupe minimum correspondante et redessiner le graphe avec le flot maximum.

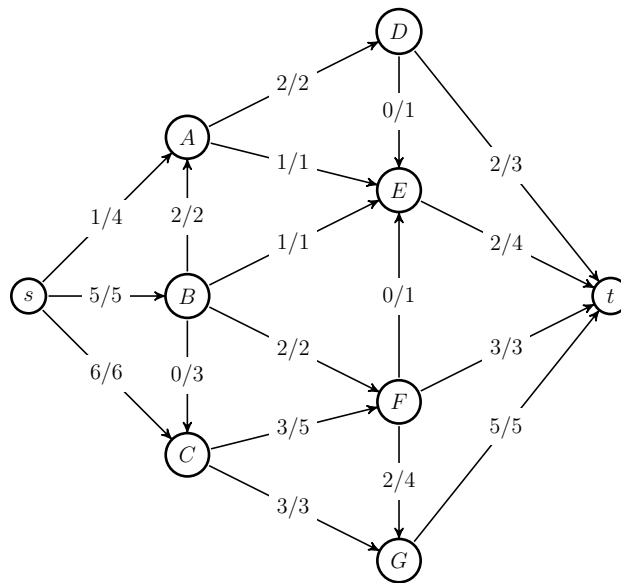


FIGURE 3 – Réseau

Rappel :

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 5 FlotMax(G, c, s, t)

```
1: pour tout  $e$  de  $E(G)$  faire
2:    $f[e] \leftarrow 0$ 
3: fin pour
4: répéter
5:   Marquage( $G, c, f, s, t$ )
6:   si  $t \in Y$  alors
7:      $v \leftarrow t$ 
8:      $C^+ \leftarrow \{(t, s)\}$ 
9:      $C^- \leftarrow \emptyset$ 
10:    tant que  $v \neq s$  faire
11:       $e \leftarrow A[v]$ 
12:      si  $v = T[e]$  alors
13:         $C^+ \leftarrow C^+ \cup \{e\}$ 
14:         $v \leftarrow I[e]$ 
15:      sinon
16:         $C^- \leftarrow C^- \cup \{e\}$ 
17:         $v \leftarrow T[e]$ 
18:      fin si
19:    fin tant que
20:  fin si
21:  pour tout  $e \in C^+$  faire
22:     $f(e) \leftarrow f(e) + \delta[t]$ 
23:  fin pour
24:  pour tout  $e \in C^-$  faire
25:     $f(e) \leftarrow f(e) - \delta[t]$ 
26:  fin pour
27: jusqu'à  $t \notin Y$ 
```

Algorithme 6 Marquage(G, c, f, s, t)

```
1:  $Y \leftarrow \{s\}$ 
2:  $\delta(s) \leftarrow +\infty$ 
3:  $Max \leftarrow \text{faux}$ 
4: tant que  $t \notin Y$  et  $Max = \text{faux}$  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:       $Max \leftarrow \text{vrai}$ 
16:    fin si
17:  fin si
18: fin tant que
```

4 Utilisation de Dijkstra dans un graphe avec un arc de poids négatif

Soit G un graphe orienté et valué par une fonction w ayant un et un seul arc de poids négatif. On note α l'origine et β l'extrémité de cet arc, *i.e.* $w(\alpha, \beta) < 0$.

Dans la suite on notera $d_s(G)$ les distances des plus courts chemins issu du sommet s vers tous les autres sommets de G . On notera $d_s(G, u)$ la distance d'un plus court chemin issu du sommet s vers le sommet u dans G .

4.1) En utilisant l'algorithme de *Dijkstra*, rappelé ci-dessous, trouver une façon de détecter l'existence d'un circuit dans G dont la somme des poids des arcs est négative (un tel circuit sera appelé un **circuit absorbant**).

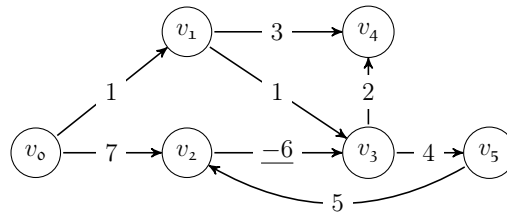
Dans la suite on suppose que le graphe G ne possède pas de circuit absorbant, mais qu'il possède un arc (α, β) tel que $w(\alpha, \beta) < 0$. Pour chaque sommet de G , on veut savoir si un plus court chemin depuis un sommet distingué s doit emprunter l'arc de poids négatif. Pour ce faire, on procède en deux étapes :

- i.* On applique l'algorithme de *Dijkstra* au graphe G' obtenu en supprimant de G l'arc de poids négatif et on calcule la distance $d_s(G')$ entre s et tous les sommets de G' .

ii. Soit G'' le graphe ayant comme sommets ceux qui dans G' sont accessibles depuis β et tous les arcs de G les reliant excepté l'arc (α, β) de poids négatif. On calcule, en appliquant l'algorithme de *Dijkstra*, la distance $d_\beta(G'')$ entre β et tous les sommets de G'' .

4.2) Comment utiliser les distances ainsi calculées pour savoir si le plus court chemin de s à un sommet u quelconque est l'un de ceux qui contiennent l'arc de poids négatif?

4.3) Appliquer votre algorithme au graphe G ci-dessous en utilisant le sommet $s = v_0$ comme sommet distingué.



Rappel :

```

0  Dijkstra(G,w,s)
1  pour chaque sommet v de V(G) faire
2    d[v] <- infini
3    pere[v] <- nil
4    couleur[v] <- BLANC
5  d[s] <- 0
6  F <- FILE_PRIORITE(V(G), d)
7  tant que F non vide faire
8    pivot <- EXTRAIRE_MIN(F)
9    couleur[pivot] <- GRIS
10   pour tout arc e = (pivot, v) arc sortant de pivot faire
11     si couleur[v] = BLANC et d[v] > d[pivot] + w(e)
12       alors d[v] <- d[pivot] + w(e)
13       pere[v] <- pivot
14   couleur[pivot] <- NOIR

```