Examen d'algorithmique

Jeudi 20 janvier 2020 9h30 – 11h30 / Aucun document autorisé

Mode d'emploi : Le barème est donné à titre indicatif. **La qualité de la rédaction sera très fortement prise en compte pour la note.** On peut toujours supposer une question résolue et passer à la suite. On peut utiliser sans les décrire les algorithmes du poly (l'annexe contient l'algorithme de Dijkstra, celui de Bellman-Ford et celui de Floyd).

Exercice 1 : Arbres couvrants minimaux – (4 points)

1. On considère le graphe non orienté valué G_2 de la figure 1 ci-dessous. Donner deux ACM distincts pour ce graphe.
2. Selon l'ordre dans lequel les arêtes de même poids sont examinées, l'algorithme de Kruskal ne retourne pas le même ACM. Pour un ACM donné, existe-t-il nécessairement un ordre tel que cet arbre soit la sortie de l'algorithme de Kruskal ?

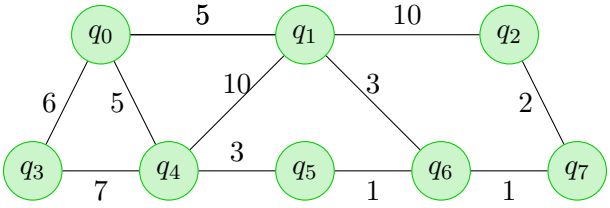


FIGURE 1 – Le graphe G_2.

Exercice 2 : PCC et routage – (4 points)

On considère un problème de routage dans un réseau fixe décrit par un graphe non-orienté valué $G = (S, A, w)$ avec $w : A \to \mathbb{N}$ où $w(u, v)$ représente la longueur de l'arc (u, v) (c'est-à-dire le temps de transmission d'un message de u vers v). Chaque noeud représente un routeur qui doit dispatcher les messages qu'il reçoit vers le bon destinataire : lorsque le noeud x reçoit un message pour le noeud y, alors si $x = y$ c'est fini, et sinon, il doit l'envoyer à l'un de ses voisins z afin que z transmette à son tour le message à un voisin, *etc.* Bien sûr on souhaite utiliser des PCC : si x envoie le message pour y à son voisin z, c'est qu'il existe un PCC reliant x à y dont la première arête est (x, z).

Proposer un algorithme qui construit pour chaque noeud x de G, une table des distances $\mathsf{d}_x[-]$ donnant les distances des PCC de x aux autres noeuds (ainsi $\mathsf{d}_x[y]$ est égal à la distance d'un PCC de x à y) et une table de routage $\mathsf{r}_x[-]$ indiquant à quel voisin de x il faut envoyer les messages selon leur destinataire ($\mathsf{r}_x[y] = z$ dans notre exemple). Donner la complexité de votre algorithme.

Exercice 3 : Graphes pondérés – (3 points)

On dispose d'un graphe orienté valué $G = (S, A, w)$ où la fonction $w : S \to \mathbb{R}^+$ associe une valeur réelle positive aux **sommets** (et non aux arcs). La valeur $w(s)$ représente un coût à payer (ou une distance) lorsqu'on arrive dans le sommet s. Ainsi le chemin $s \to s' \to s''$ aura un coût égal à $w(s') + w(s'')$, et plus généralement un chemin $q_1 \to q_2 \to \ldots \to q_n$ aura un coût $\sum_{i=2\ldots n} w(q_i)$. Proposer une méthode pour trouver les coûts minimum pour atteindre chaque sommet depuis un sommet de départ s donné. Appliquer la sur le graphe G_3 de la figure 2 depuis le sommet q_0.

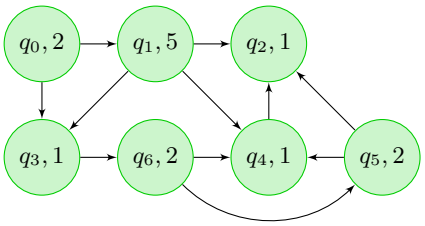


FIGURE 2 – Le graphe pondéré G_4

Exercice 4 : PCC et graphes acyclique – (5 points)

On considère un graphe orienté valué $G = (S, A, w)$ avec $w : A \to$ **R sans circuit**.

Proposer un algorithme efficace pour trouver les distances des PCC depuis un sommet q_0 donné (on supposera qu'aucun arc arrive en q_0, c'est-à-dire $\deg^+(q_0) = 0$). Justifier votre algorithme.

Appliquer votre algorithme sur le graphe G_1 de la figure 3 ci-contre.

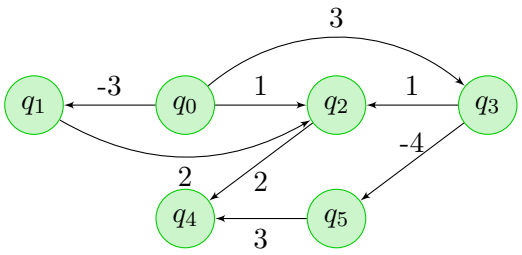


FIGURE 3 – Graphe G_1.

Indices : on vise un algorithme de complexité en $O(|S| + |A|)$. On pourra utiliser un tri topologique pour énumérer les sommets accessibles depuis s de manière à ce que l'énumération $q_0\, q_1\, q_2 \ldots q_n$ vérifie : $i < j \;\Rightarrow\; (q_j, q_i) \notin A$.

Exercice 5 : Flots – (4 points)

On considère le réseau de transport R de la figure 4 :

1. Appliquer l'algorithme de Ford-Fulkerson pour trouver un flot maximal. On donnera les flots intermédiaires et les graphes des augmentations obtenus par l'algorithme (une feuille est préremplie, voir pages 5 et 6 du sujet). Attention : **on demande à ce que votre application de l'algorithme utilise au moins un arc arrière**.
2. En déduire une coupe de capacité minimale du réseau R.

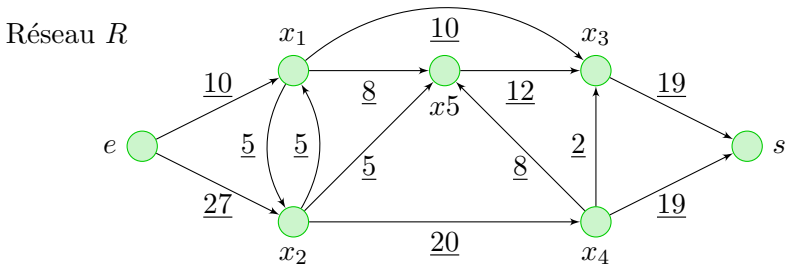


FIGURE 4 – Le réseau de transport R pour l'exercice 6.

Annexe

On rappelle ci-dessous l'algorithme de Dijkstra :

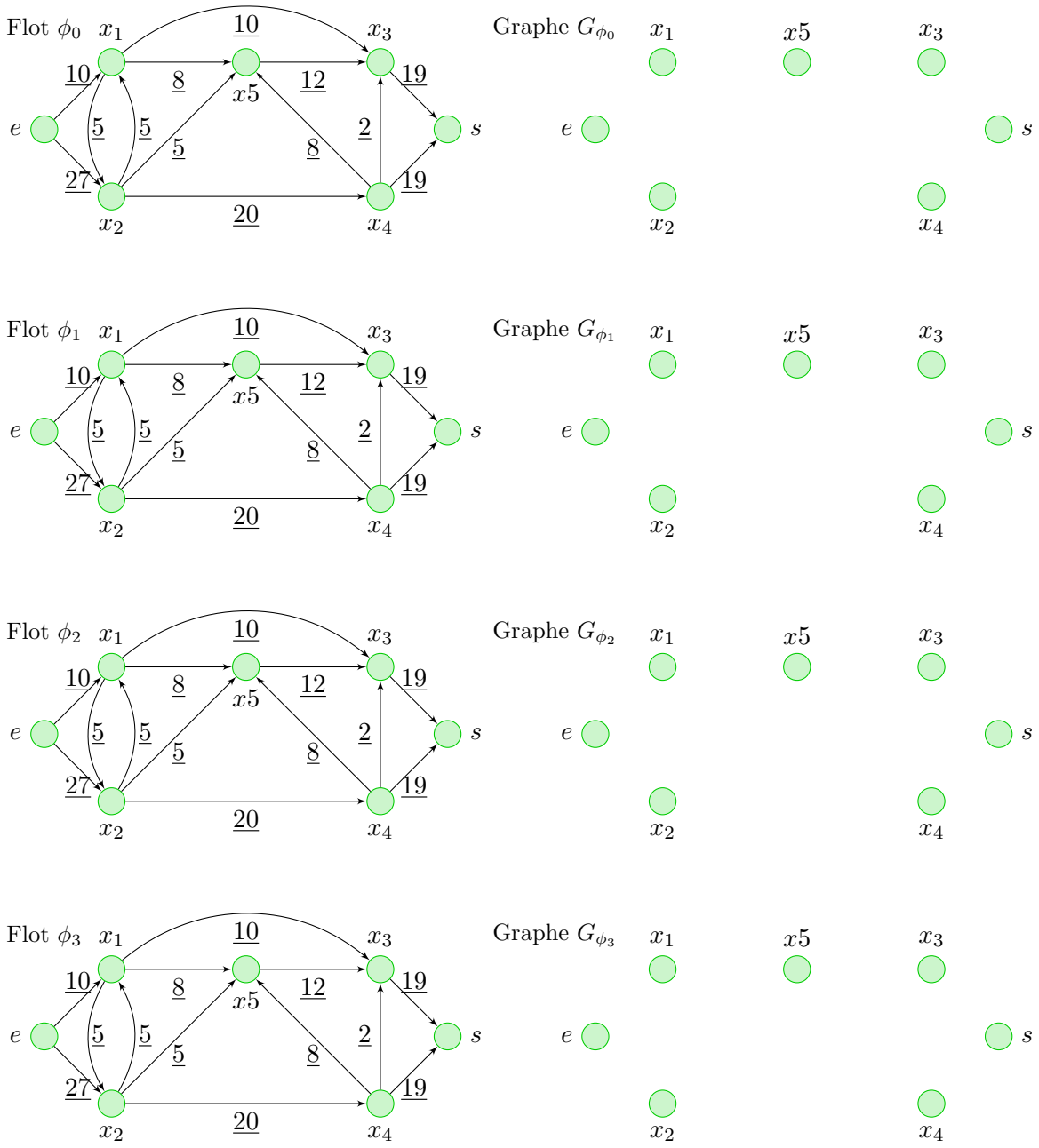
Procédure `PCC-Dijkstra`(G, s)
`//`$G = (S, A, w)$ `: un graphe orienté valué`
`//`$s \in S$ `: un sommet origine.`
begin
  pour chaque $u \in S$ **faire**
    $\Pi[u] :=$ nil
    $\mathsf{d}[u] := \begin{cases} 0 & \text{si } u = s \\ \infty & \text{sinon} \end{cases}$
  $F :=$ FilePriorité(S, d)
  tant que $F \neq \emptyset$ **faire**
    $u :=$ Extraire-Min(F)
    pour chaque $(u, v) \in A$ **faire**
      si $d[v] > d[u] + w(u, v)$ **alors**
        $\mathsf{d}[v] := \mathsf{d}[u] + w(u, v)$
        $\Pi[v] := u$
        MaJ-F-Dijkstra(F, d, v)
  return (d, Π)
end

On rappelle ci-dessous l'algorithme de Floyd-Warshall et l'algorithme de Bellman-Ford :

Procédure `PCC-Floyd`(G)
`//`$G = (S, A, w)$ `: orienté valué`
`//avec` $S = \{x_1, \ldots, x_n\}$
`//avec` $M = (\alpha_{ij})_{1 \leq i,j \leq n}$ `la`
`//matrice correspondant à` A
begin
  `//On initialise` D `avec` M`:`
  pour $i = 1 \ldots n$ **faire**
    pour $j = 1 \ldots n$ **faire**
      $d_{ij} := \alpha_{ij}$
      si $\alpha_{ij} \neq \infty$ **alors** $\pi_{ij} := i$
  pour $k = 1 \ldots n$ **faire**
    pour $i = 1 \ldots n$ **faire**
      pour $j = 1 \ldots n$ **faire**
        si $d_{ij} > d_{ik} + d_{kj}$ **alors**
          $d_{ij} := d_{ik} + d_{kj}$
          $\pi_{ij} := \pi_{kj}$
  return D, Π
end

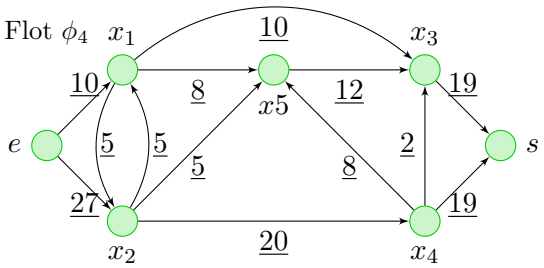
Procédure `PCC-Bellman-Ford`(G, s)
`//`$G = (S, A, w)$ `: un graphe orienté valué`
`//`$s \in S$`: un sommet origine`
begin
  pour chaque $u \in S$ **faire**
    $\Pi[u] :=$ nil
    $\mathsf{d}[u] := \begin{cases} 0 & \text{si } u = s \\ \infty & \text{sinon} \end{cases}$
  pour $i = 1$ *à* $|S| - 1$ **faire**
    pour chaque $(u, v) \in A$ **faire**
      si $d[v] > d[u] + w(u, v)$ **alors**
        $\mathsf{d}[v] := \mathsf{d}[u] + w(u, v)$
        $\Pi[v] := u$
  pour chaque $(u, v) \in A$ **faire**
    si $d[v] > d[u] + w(u, v)$ **alors**
      return $(\bot, -, -)$
  return $(\top, d, \Pi)$
end

A détacher et à remettre avec la copie.

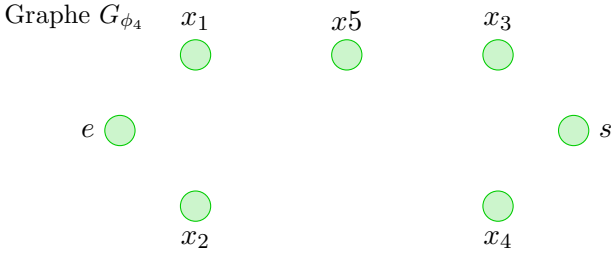


A détacher et à remettre avec la copie.

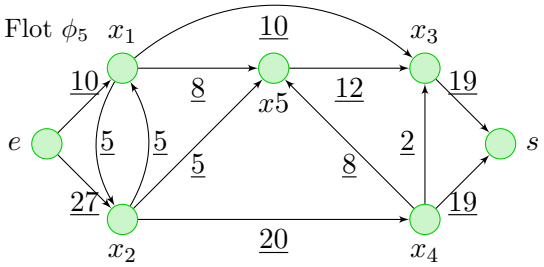
Flot ϕ_4



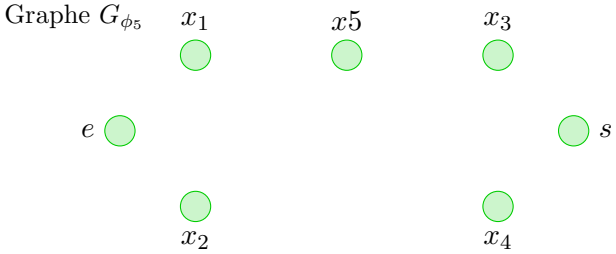
Graphe G_{ϕ_4}



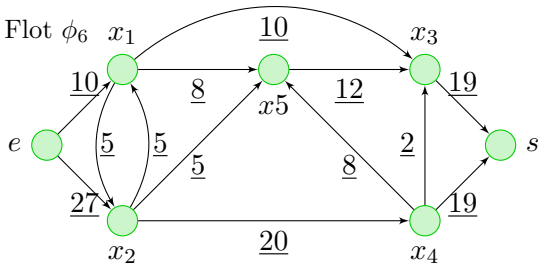
Flot ϕ_5



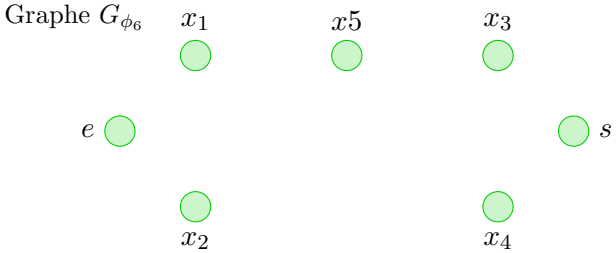
Graphe G_{ϕ_5}



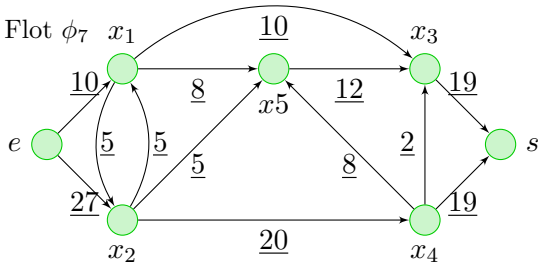
Flot ϕ_6



Graphe G_{ϕ_6}



Flot ϕ_7



Graphe G_{ϕ_7}