CentraleSupélec

Algorithmique et complexité

TD 2/7 – Des chemins et des arbres

Ce TD est l'occasion de s'exercer à la résolution de problèmes à base de graphes et d'approfondir la compréhension des problèmes traités en cours. Notamment le problème du plus court chemin et celui de l'arbre couvrant de poids minimal.

Exercice 1 : Quelques scènes de la vie campagnarde

Un hameau de huit maisons A, B, C, D, E, F, G et H est relié par un réseau routier dont la structure simplifiée est définie par la matrice suivante :

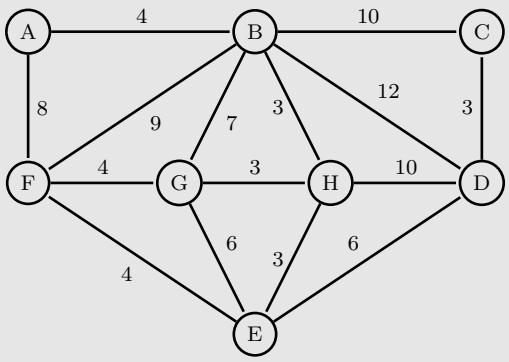
	A	B	C	D	E	F	G	H
A	0	4	∞	∞	∞	8	∞	∞
B	4	0	10	12	∞	9	7	3
C	∞	10	0	3	∞	∞	∞	∞
D	∞	12	3	0	6	∞	∞	10
E	∞	∞	∞	6	0	4	6	3
F	8	9	∞	∞	4	0	4	∞
G	∞	7	∞	∞	6	4	0	3
H	∞	3	∞	10	3	∞	3	0

Dans cette matrice $M(i, j)$ représente le temps de parcours direct de i à j en minutes. Bien sûr, $M(i, i) = 0$ et $M(i, j) = \infty$ signifie qu'il n'y pas de route reliant directement i à j.

Question 1

Faire une représentation graphique dans le plan du graphe non orienté correspondant. Une représentation planaire (sans intersection) est possible.

Élements de correction :



Question 2

Un médecin de campagne habitant en **G** veut déterminer, en cas d'urgence médicale, les itinéraires les plus rapides le reliant à chacun des autres lieux du hameau. Calculer ces itinéraires : on définira :

— la nature de ce problème,
— la méthode employée,
— les détails de chacune des étapes.

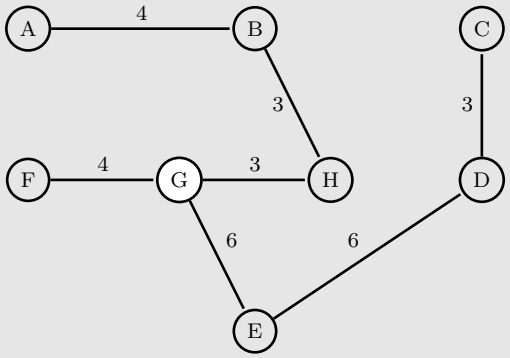
Faites une représentation du sous réseau (graphe partiel) correspondant aux itinéraires obtenus.

Élements de correction :

La frontière correspond aux cases grisées et le noeud en cours de visite est en gras. La première colonne contient l'évolution des sommets visités.

	A	B	C	D	E	F	H
$V = \{G\}$	∞	7,G	∞	∞	6,G	4,G	**3,G**
$V = \{G, \mathbf{H}\}$	∞	6,H	∞	13,H	6,G	**4,G**	-
$V = \{G, H, \mathbf{F}\}$	12,F	**6,H**	∞	13,H	6,G	-	-
$V = \{G, H, F, \mathbf{B}\}$	10,B	-	16,B	13,H	**6,G**	-	-
$V = \{G, H, F, B, \mathbf{E}\}$	**10,B**	-	16,B	12,E	-	-	-
$V = \{G, H, F, B, E, \mathbf{A}\}$	-	-	16,B	**12,E**	-	-	-
$V = \{G, H, F, B, E, A, \mathbf{D}\}$	-	-	15,D	-	-	-	-
$V = \{G, H, F, B, E, A, D, \mathbf{C}\}$	**10,B**	**6,H**	**15,D**	**12,E**	**6,G**	**4,G**	**3,G**

L'arbre correspondant aux plus courts chemins est le suivant :



Supposons maintenant que les habitants souhaitent renouveler l'alimentation en eau des huit maisons à partir d'un château d'eau situé en **H** par un réseau enfoui qui doit suivre certaines des routes existantes. Chaque nouvelle canalisation a un coût proportionnel à la longueur de la route correspondante, donc à la durée de son parcours. Pour cela, on veut construire un réseau d'alimentation en eau de coût minimal.

Question 3

Quelle est la structure du graphe partiel recherché ? Comment construire un tel réseau ?

Élements de correction :

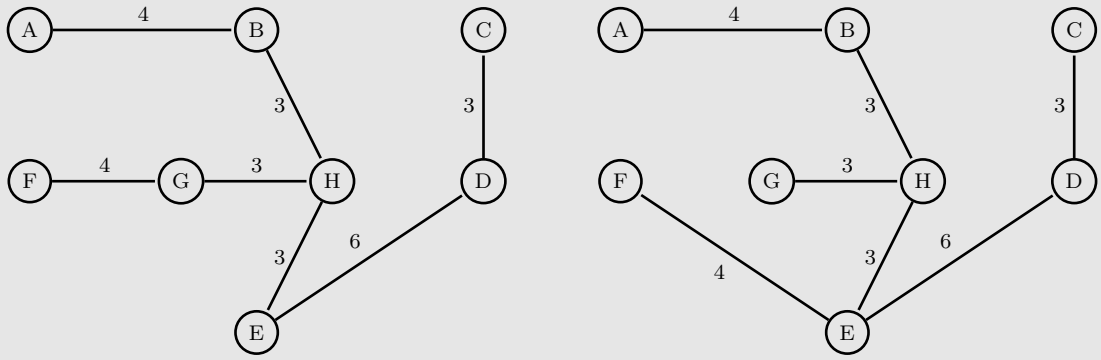
En appliquant l'algorithme de Kruskal, les arêtes sont considérées dans l'ordre suivant : BH, CD, EH, GH, AB, EF (ou bien FG mais pas les deux car cela crée un circuit) et enfin DE.

En appliquant l'algorithme de Prim, si on commence depuis H, les arêtes sont considérés dans l'ordre suivant : BH (dist(B)=3), HG(dist(G)=3), HE(dist(E)=3), AB(dist(A)=4), FG(dist(F)=4) (ou bien EF selon si parent F est G ou bien E), ED(dist(D)=6), CD(dist(D)=3).

Question 4

Dessinez le réseau obtenu. Le lieu où est situé le château d'eau a-t-il un impact (structure, coût) sur le réseau obtenu ?

Élements de correction :



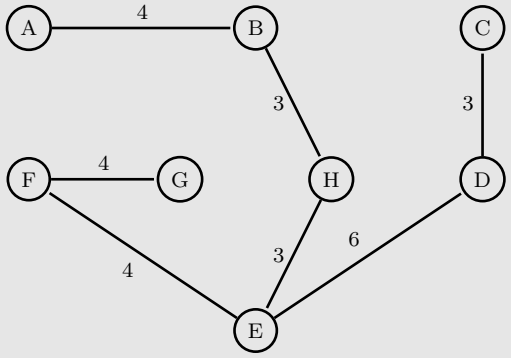
Deux arbres couvrants minimums possibles selon l'algorithme utilisé. Leurs coûts sont identiques de 26. Les arbres obtenus sont indépendants du lieu où est situé le château d'eau. Notons qu'ils sont différents de l'arbre des plus courts chemins qui a un coût global de 29.

Question 5

Des intempéries rendent la liaison GH inutilisable. Que deviennent alors les résultats des questions précédentes (question 2 et question 4) ?

Élements de correction :

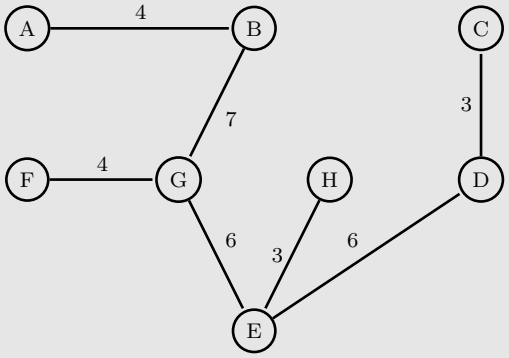
1. En effet, l'arbre couvrant minimal se déconnecte en deux sous arbres. par exemple, si on considère le premier MST (à gauche) de la question 4, celui-ci se sépare en 2 sous-arbres contenant respectivement {F,G} et {A,B,C,D,E,H} comme sommets. Ainsi, pour obtenir la solution recherchée, il suffit de choisir, parmi les arêtes reliant ces deux sous arbres, c'est à dire dans l'ensemble {AF, BF, BG, EF, EG}, une arête de coût minimal, on prend donc EF de coût 4, d'où le nouvel arbre couvrant d'un poids de 27 :



Arbre couvrant minimum

Un exercice complémentaire consacré à la mise à jour d'arbre couvrant permet d'étudier cette problématique en détail.

2. Pour les plus courts chemins, il faudrait tout recalculer **comme l'arête appartient à l'arbre des plus courts chemins**. Attention, Il ne suffit pas de relancer l'algo sur le sous-graphe ne contenant que les nœuds déconnectés, même si les plus courts chemins des noeuds qui restent du bon côté (avec le noeud de départ) dans l'arbre coupé ne vont pas voir leur distance changer. Nous obtiendrions alors :



Plus court chemins

Exercice 2 : DAG

L'algorithme des plus courts chemins trouve les plus courts chemins dans un graphe de topologie quelconque. Pour certaines familles de graphes, il est possible d'écrire des algorithmes plus performants (en temps de calcul). Dans cet exercice, nous allons voir comment « améliorer » l'algorithme des plus courts chemins dans le cas des graphes orientés **sans circuits** (en anglais : *Directed Acyclic Graphs* ou DAG).

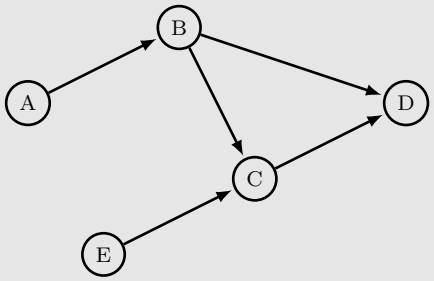
Considérons $G = (V, E)$ un DAG pondéré.

Question 1

Dessinez un graphe de ce type avec quelques sommets.

Élements de correction :

Un DAG quelconque :

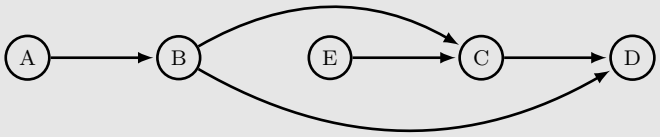


Question 2

Dessinez **le même graphe** en positionnant les sommets sur une ligne de manière que tous les arcs vont dans le même sens. Qu'en déduisez-vous ?

Élements de correction :

Une possibilité :



Il s'agit d'un **ordre topologique**. Un exercice complémentaire du TD1 traite entièrement ce sujet.

Question 3

En utilisant la propriété observée dans la question précédente, proposez une variante de l'algorithme des plus courts chemins qui calcule les plus courts chemins en $\mathcal{O}(|V| + |E|)$. Écrivez votre algorithme en python ou en pseudo-code en décrivant précisément les structures de données utilisées.

Élements de correction :

L'algorithme des plus courts chemins n'est plus nécessaire, car en parcourant le noeuds dans l'ordre topologique :

- on n'a plus besoin de garder les noeuds visités !
- on n'a plus besoin de gérer la frontière et en calculer le minimum à chaque étape !

Le traitement en ordre topologique assure qu'au moment de calculer la distance la plus courte pour un sommet v tous les chemins y menant depuis le sommet de départ s passent par les sommets dont les distances les plus courtes sont déjà définitives.

```
def PCCs_topological(graph, s)
    parent = {}
    dist = {v: math.inf for v in nodes(graph)}
    dist[s] = 0

    for v in topological_order(nodes(graph)): # parcours en ordre topologique
        for u in graph:
            if dist[u] > dist[v] +  distance(graph,v,u):
                dist[u] = dist[v] +  distance(graph,v,u)
                parent[u] = v
```

Exercice 3 : Arborescence couvrante minimale (partie avancée)

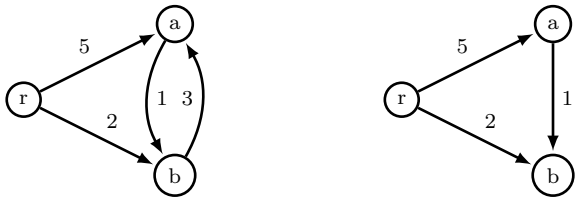
Attention : cet exercice est à réserver pour les groupes avancés.

Considérons un graphe orienté et pondéré G avec une racine r à partir de laquelle tous les sommets sont atteignables. Une arborescence couvrante de poids minimale de G à partir de r est un sous-graphe de G tel que tous les sommets sont toujours atteignables depuis r et que la somme des poids est minimale. Il s'agit de la version « orientée » d'un arbre couvrant de poids minimal vu en cours.

Considérons $G = (V, E)$ un graphe orienté et pondéré avec une racine r.

Question 1

Peut-on appliquer l'algorithme de Kruskal et l'algorithme de Prim sur G pour trouver une arborescence couvrante de poids minimal ? Si oui, expliquez pourquoi. Sinon, donnez un contre-exemple. Vous pouvez justifier votre réponse en utilisant les deux graphes suivants avec la racine r.



Élements de correction :

Pour le graphe à gauche, nous avons trois arborescences couvrantes à partir de r.
- $r \to a \to b$, dont le poids est 6
- $r \to b \to a$, dont le poids est 5
- $r \to b$ et $r \to a$, dont le poids est 7

L'arborescence couvrante de poids minimal pour ce graphe est donc $r \to b \to a$ avec le poids 5.

Pour le graphe à droite, on a deux arborescences couvrantes à partir de r.
- $r \to a \to b$, dont le poids est 6
- $r \to b$ et $r \to a$, dont le poids est 7

L'arborescence couvrante de poids minimal pour ce graphe est donc $r \to a \to b$ avec le poids 6.

On ne peut pas appliquer l'algorithme de Kruskal et l'algorithme de Prim sur un graphe orienté.
- l'algorithme de Kruskal : avec Kruskal, pour le graphe à gauche, $a \to b$ est le premier arc à choisir, qui ne fait pas partie de l'arborescence couvrante minimale.
- l'algorithme de Prim : avec Prim à partir de r, pour le graphe à droite, l'arc $r \to b$ est d'abord choisi avant de prendre l'arc $r \to a$, ce qui n'est pas l'arborescence couvrante minimale pour ce graphe.

Nous allons maintenant étudier un autre algorithme pour trouver une arborescence couvrante de poids minimal.

Les étapes principales sont décrites ci-dessous :

1. supprimez tous les arcs dans le graphe G dont la destination est la racine r
2. pour chaque sommet v autre que la racine, prenez l'arc de poids le plus faible dont la destination est v, noté (u, v) pour lequel $min_u \omega(u, v)$
3. Considérez l'ensemble d'arcs choisis $S = \{(u, v) \mid v \in V \setminus \{r\} \wedge min_u \omega(u, v)\}$. Si S ne contient pas de cycles, alors S est une arborescence couvrante de poids minimal. Sinon, pour chaque cycle dans S, noté C, il faudrait le traiter comme suit.
   (a) pour chaque arc $(i, j)(i \in V \setminus C, j \in C)$ (un arc entrant dans le cycle), modifiez son poids : $\omega'(i, j) = \omega(i, j) - \omega(x(j), j)$, où $\omega(x(j), j)$ est le poids de l'arc dont la destination est j dans C.
   (b) ajoutez l'arc entrant dans le cycle $(i, j) \in E$ dont le poids modifié $\omega'(i, j)$ est le plus faible parmi tous les arcs entrant dans le cycle
   (c) enlevez l'arc dans C dont la destination est j

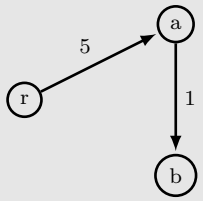
Question 2

Appliquez ce nouvel algorithme sur les graphes ci-dessus et donnez le résultat pour chaque étape.

Élements de correction :

— Pour le graphe à droit, on donne le résultat pour chaque étape.

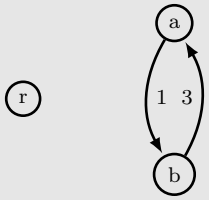
1. comme il n'y a pas d'arc dont la destination est r, alors on ne fait rien.
2. concernant chaque sommet autre que r : pour a, on prend l'arc $r \to a$ car il n'y a qu'un arc dont la destination est a ; pour b, on prend l'arc le plus faible dont la destination est b, donc $a \to b$.



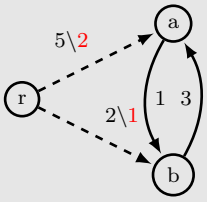
3. comme maintenant, l'ensemble d'arcs choisis ($r \to a$ et $a \to b$) ne contiennent pas un cycle, donc il s'agit bien de l'arborescence couvrante minimale (voir la réponse pour la question précédente.)

— Pour le graphe à gauche :

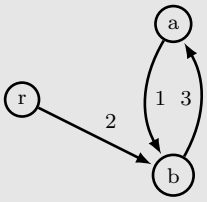
1. Il n'y a pas d'arc dont la destination est r, alors on ne fait rien.
2. concernant chaque sommet autre que r : pour a, on prend l'arc $b \to a$ car son poids est le plus faible dont la destination est a ; pour b, on prend l'arc le plus faible dont la destination est b, donc $a \to b$. Alors on obtient



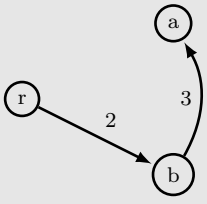
3. l'ensemble d'arcs choisis ($a \to b$ et $b \to a$) contiennent bien un cycle $C = a \to b \to a$, donc on continue à traiter ce cycle comme suit.
   (a) Appliquez le formule ci-dessus, nous avons $\omega'(r,a) = \omega(r,a) - \omega(b,a) = 5 - 3 = 2$ et $\omega'(r,b) = \omega(r,b) - \omega(a,b) = 2 - 1 = 1$. Maintenant nous avons le graphe suivant, où le poids rouge est le poids modifié.



   (b) l'arc entrant dans ce cycle dont le poids modifié est le plus faible est donc $r \to b$. On ajoute cet arc original



   (c) l'arc dans C dont la destination est b est bien $a \to b$, donc il faut le supprimer.



   Il n'y a plus de cycle dans le graphe, donc il s'agit bien de l'arborescence couvrante minimale (voir la réponse pour la question précédente).

Question 3

Maintenant vous comprenez chaque étape de l'algorithme, implémentez-le en python et teste-le à l'aide du code suivant. Votre fonction principale est $acm(root, graphe)$ avec $graphe$ représenté par un dictionnaire d'adjacence :

```
graphe={"r":{"a":1,"b":10},
            "a":{"b":8,"e":2},
            "b":{"d":6},
            "c":{"b":4,"a":7},
            "d":{"c":3},
            "e":{"c":9,"d":11}
    }
h = acm("r",graphe)
assert h == {'r': {'a': 1}, 'c': {}, 'a': {'e': 2, 'b': 8}, 'b': {'d': 6}, 'd': {'c': 3}, 'e': {}}
```

Élements de correction :