TD 6 - Graphes orientés et plus courts chemins.

Exercice 1.

Définition : Un *ordre topologique* est un ordre total sur les sommets d'un graphe qui étend l'ordre partiel donné par les arcs. En d'autres termes, pour tout arc (u, v) du graphe, u doit apparaître avant v dans l'ordre topologique.

Un algorithme de tri topologique est un algorithme qui ordonne les sommets d'un graphe suivant un ordre topologique. Ces algorithmes sont utiles lorsqu'on a des graphes de contraintes d'ordonnancement sur des tâches pour décider d'un ordre d'exécution de ces tâches qui respecte les contraintes de précédence.

(*Remarque* : on peut utiliser les algorithmes de tri topologique du graphe de précédence d'un ensemble de transactions pour obtenir un ordre de sérialisation (lorsque l'ensemble le permet).)

1. Quels sont les graphes pour lesquels il existe un ordre topologique ?
2. Quels sont les graphes pour lesquels l'ordre topologique est unique ?
3. Quels sont les graphes pour lesquels n'importe quel ordre de leurs sommets est un ordre topologique ?

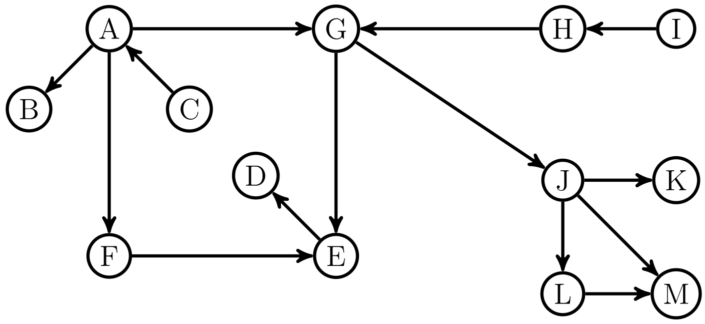
   Vous trouverez ci-dessous l'algorithme de Kahn, qui renvoie un ordre topologique si le graphe donné en admet un.

Algorithme 1 : KAHN(G)

Entrées : un graphe orienté acyclique G.
Sortie : les sommets de G ordonnés topologiquement.

```
/* stocker les degrés entrants et les sources                    */
résultat ← liste();
sources ← pile();
degrés_entrants ← tableau(G.nombre_sommets(), 0);
pour chaque v ∈ G.sommets() faire
    degrés_entrants[v] ← G.degré_entrant(v);
    si degrés_entrants[v] = 0 alors sources.empiler(v);
/* dépiler les sources, les ajouter au résultat, et empiler les
   nouvelles sources                                             */
tant que sources.pas_vide() faire
    u ← sources.dépiler();
    résultat.ajouter_en_fin(u);
    pour chaque v ∈ G.successeurs(u) faire
        degrés_entrants[v] ← degrés_entrants[v] −1;
        si degrés_entrants[v] = 0 alors sources.empiler(v);
renvoyer résultat;
```

4. Appliquez cet algorithme au graphe acyclique ci-dessous.

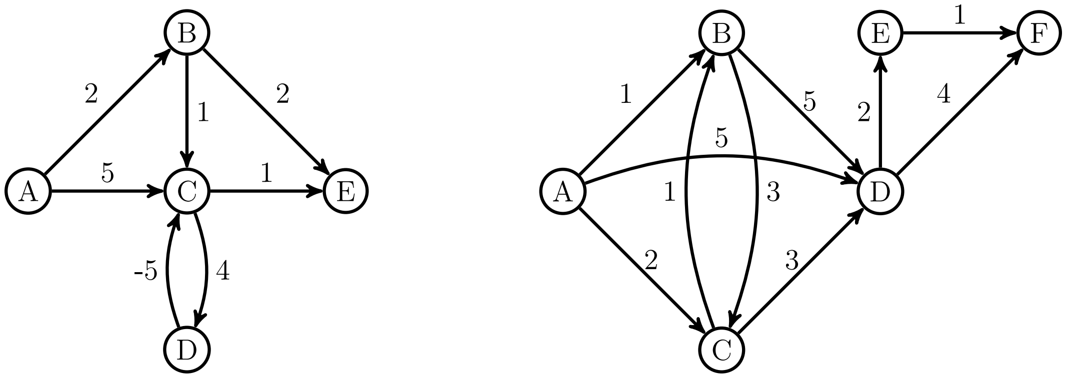


5. Expliquez en quelques phrases le principe de cet algorithme.

Exercice 2.

Certaines des questions ci-dessous vous demandent de calculer un plus court chemin. Vous pouvez généralement le calculer simplement à la main, sans donner le déroulement de l'un des algorithmes vus au cours.

1. Rappelez la définition d'un plus court chemin entre deux sommets s et t d'un graphe G.
2. Voici deux graphes :



Dans le premier graphe, quel est le plus court chemin du sommet A au sommet E ?

3. Recalculez ce plus court chemin en remplaçant l'arc de poids -5 par un arc de poids 2.
4. Calculez un plus court chemin du sommet A au sommet F dans le second graphe.
5. Montrez la propriété de découpage suivante :

   Soit $G = (V, A)$ un graphe pondéré sans cycle de poids négatif. Si $P = (s_1, s_2, \ldots, s_n)$ est un plus court chemin dans G de s_1 à s_n, alors pour tout $i \in \{2, 3, \ldots, n-1\}$, le chemin $s_1, s_2, \ldots, s_i$ est un plus court chemin dans G de s_1 à s_i et $s_i, s_{i+1}, \ldots, s_n$ est un plus court chemin dans G de s_i à s_n.

   Autrement dit, si on coupe un plus court chemin en deux, on obtient deux plus courts chemins.
6. Montrez la propriété suivante :

   Soit $G = (V, A)$ un graphe pondéré orienté sans cycle de poids négatif. Si le sommet t est accessible dans G à partir du sommet s, alors il existe un plus court chemin de s à t contenant au plus $|V| - 1$ arcs.

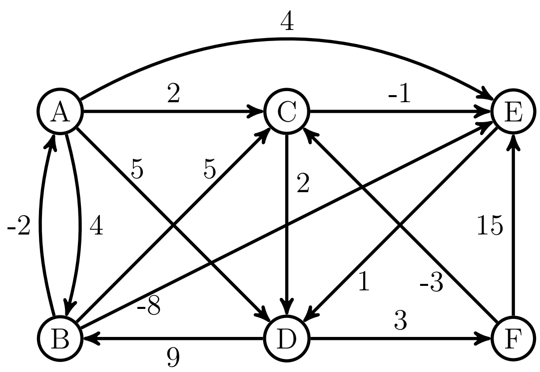
Exercice 3.

L'objectif de cet exercice est de montrer que l'algorithme de Bellman-Ford décrit en pseudocode ci-dessous permet de construire, lorsqu'il existe, un arbre des plus courts chemins au départ du sommet s dans le graphe G.

Algorithme 2 : BellmanFord(G, s)

d $\leftarrow$ tableau(G.nombre_sommets(), $+\infty$);
d[s] $\leftarrow$ 0;
p $\leftarrow$ tableau(G.nombre_sommets(), NIL);
pour *i allant de* 1 *à G.nombre_sommets()* -1 **faire**
    pour chaque $(u, v, x) \in G.arcs()$ **faire**
        si *d[v]* $>$ *d[u]* $+ x$ **alors**
            d[v] $\leftarrow$ d[u] $+ x$;
            p[v] $\leftarrow u$;
pour chaque $(u, v, x) \in G.arcs()$ **faire**
    si *d[v]* $>$ *d[u]* $+ x$ **alors renvoyer** NIL;
renvoyer (d, p);

1. Utilisez l'algorithme de Bellman-Ford sur le graphe ci-dessous pour obtenir les plus courts chemins à partir du sommet A, puis à partir du sommet E. Ne donnez le détail que pour la première étape ; pour les étapes suivantes, on se contentera de donner le résultat final du tableau de distances.



2. Remplacez le poids de l'arc $D \to B$ par 6 et relancez l'algorithme en partant de A.
3. A quoi servent les tableaux d et p ?
4. A quoi sert la dernière boucle ?
5. Dans le cas où le graphe G n'a pas de cycle de poids négatif, prouvez l'invariant :

   Après k tours dans la boucle principale, pour tout sommet t de G, s'il existe un plus court chemin de s à t ayant au plus k arcs alors `d[t]` est égal au poids de ce plus court chemin ; sinon, `d[t]` est supérieur ou égal au poids du plus court chemin de s à t.
6. Quelle est la complexité de cet algorithme ?
7. Proposez une amélioration simple de cet algorithme.

Exercice 4.

On souhaite convertir de l'argent d'une devise dans une autre. Toutes les conversions ne sont pas possibles : pour deux devises A et B, on ne peut pas toujours trouver une banque qui propose de changer de l'argent de A en B. On considère donc un graphe de change entre devises donnant les conversions proposées par les établissements bancaires. Ce graphe est orienté (parfois on peut convertir A en B mais pas B en A). On pondère ce graphe par la fonction de change c qui donne lorsqu'il existe le meilleur taux de change d'une devise vers une autre.

Plus précisément, on considère le graphe orienté $G = (V, A, c)$ tel que :

— les sommets représentent les différentes devises ;
— $(u, v) \in A$ signifie qu'un établissement propose le change de la devise u en la devise v ;
— pour tout arc $(u, v) \in E$, son poids $c(u, v)$ est strictement positif et correspond au meilleur taux de change de la devise u vers la devise v.

Ainsi, si on possède une quantité S dans la devise u, on peut obtenir $S.c(u, v)$ unités dans la devise v.

1. À quelle condition (exprimée sur G) peut-on changer des devises u dans des devises v ?
2. Une *séquence de change* est un chemin dans le graphe G qui correspond à la conversion d'une devise A_1 en une devise A_k en passant par les devises intermédiaires $A_2, \ldots, A_{k-1}$.

   Quel est le taux de change de A_1 en A_k dans une séquence de change $A_1, A_2, \ldots, A_k$? Si on a une quantité S dans la devise A_1, combien obtiendrez-vous d'argent dans la devise A_k via cette séquence ?
3. Dans quelle condition (exprimée sur G) quelqu'un peut-il devenir infiniment riche en changeant de l'argent ?
4. Supposons que l'on connaisse une séquence de change S_1 de la devise A en la devise B, d'une part, et une séquence S_2 de la devise A en la devise C d'autre part. Supposons que l'arc (C, B) existe dans le graphe de change. Écrivez une condition comparant les taux des séquences S_1 d'une part, de S_2 suivie de (C, B) d'autre part, et gardant la meilleure.
5. En utilisant la condition de la question précédente, écrivez une version modifiée de l'algorithme de Bellman-Ford qui calcule les meilleurs taux de change d'une monnaie A vers toutes les autres lorsque c'est possible. Pourquoi est-ce correct ?

   Indication : $\log(ab) = \log(a) + \log(b)$
6. Même question avec Dijkstra : est-ce que ça marche ?
7. ★★★ Peut-on modifier Dijkstra ou Bellman-Ford pour obtenir un arbre des plus longs chemins ?