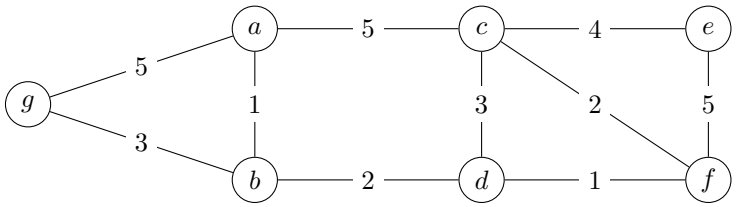
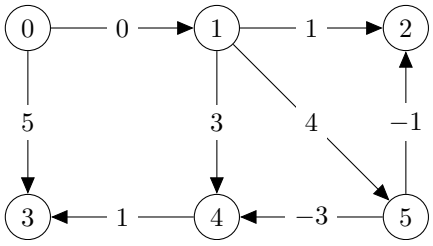
Exercice 1. Application des algorithmes

1. Appliquer l'algorithme de Dijkstra à la main sur le graphe ci-dessous pour déterminer les distances de a aux autres sommets.



2. Appliquer l'algorithme de Bellman-Ford pour trouver les distances depuis a vers n'importe quel autre sommet :



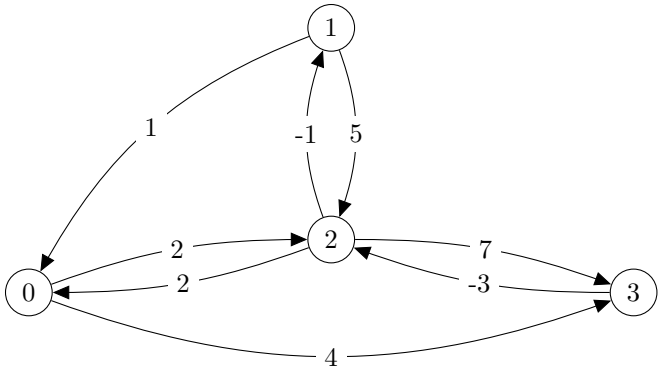
Solution : Fait en TD.

3. Appliquer l'algorithme de Floyd-Warshall au graphe représenté par la matrice d'adjacence pondérée ci-dessous, pour trouver la matrice des distances.

$$\begin{pmatrix} 0 & \infty & 2 & 4 \\ 1 & 0 & 5 & \infty \\ 2 & -1 & 0 & 7 \\ \infty & \infty & -3 & 0 \end{pmatrix}$$

► Pour calculer la $k$ème matrice des distances d_k, on regarde si on peut diminuer chaque case `d.(u).(v)` par `d.(u).(k) + d.(k).(v)`. Les valeurs mises à jour sont en gras:

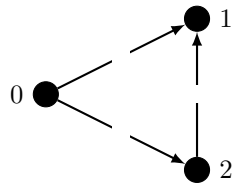
$$\left(\begin{array}{|c|ccc} \hline 0 & \infty & 2 & 4 \\ \hline 1 & 0 & \mathbf{3} & \mathbf{5} \\ 2 & -1 & 0 & \mathbf{6} \\ \infty & \infty & -3 & 0 \end{array}\right) \quad \left(\begin{array}{c|c|cc} 0 & \infty & 2 & 4 \\ \hline 1 & 0 & 3 & 5 \\ \hline \mathbf{0} & -1 & 0 & \mathbf{4} \\ \infty & \infty & -3 & 0 \end{array}\right) \quad \left(\begin{array}{cc|c|c} 0 & \mathbf{1} & 2 & 4 \\ 1 & 0 & 3 & 5 \\ \hline 0 & -1 & 0 & 4 \\ \hline \mathbf{-3} & \mathbf{-4} & -3 & 0 \end{array}\right) \quad \left(\begin{array}{ccc|c} 0 & \mathbf{0} & \mathbf{1} & 4 \\ 1 & 0 & \mathbf{2} & 5 \\ 0 & -1 & 0 & 4 \\ \hline -3 & -4 & -3 & 0 \end{array}\right)$$



Exercice 2. Petites questions

1. Trouver un exemple de graphe pour lequel Dijkstra ne fonctionne pas (ne donne pas les plus courts chemins).
2. Change t-on les plus courts chemins si on additionne/multiplie les poids de toutes les arêtes d'un graphe par une constante?
   ► Soit K une constante, $w_1(\vec{e}) = w(\vec{e}) + K$ et $w_2(\vec{e}) = w(\vec{e}) \times K$. Soit C un chemin. Alors:

- $w_1(C) = \sum_{\vec{e} \in C} (w(\vec{e}) + K) = w(C) + \ell(C)K$, où $\ell(C)$ est le nombre d'arcs de C. Tous les poids des chemins n'augmentent pas de la même façon... ce qui peut changer les plus courts chemins. Par exemple si on augmente de 2 les poids sur le graphe suivant, le plus court chemin de 0 à 1 passe de $0 \to 2 \to 1$ à $0 \to 1$:



- $w_2(C) = \sum_{\vec{e} \in C} (w(\vec{e}) \times K) = w(C) \times K$: si $K > 0$ alors minimiser $w_2(C)$ revient à minimiser $w(C)$, donc les plus courts chemins ne changent pas.

3. On suppose avoir un graphe orienté $\vec{G}$ dont les **sommets** sont pondérés par w, au lieu des arêtes. Le poids d'un chemin est alors la somme des poids de ses sommets. Comment modifier $\vec{G}$ pour pouvoir trouver des chemins de plus petit poids de $\vec{G}$, avec les algorithmes du cours?
   ▶ On remplace chaque sommet v par deux sommets v_1 et v_2 et on ajoute un arc (v_1, v_2) dont le poids est celui de v. On remplace un arc (v, v') dans $\vec{G}$ par un arc (v_2, v'_1) de poids 0. $\vec{G}$

4. Le graphe orienté $\vec{G} = (V, \vec{E})$ d'une chaîne de Markov possède une probabilité sur chaque arc. La probabilité d'un chemin est le produit des probabilités de ses arcs. Comment trouver un chemin le plus probable d'un sommet à un autre?
   ▶ 1ère solution: remplacer chaque probabilité $p(u, v)$ par $-\ln(p(u, v)) \geq 0$. Alors la somme des poids des arcs d'un chemin C est $\sum_{(u,v) \in C} -\ln(p(u, v)) = -\ln(\prod_{(u,v) \in C} p(u, v))$ donc maximiser $\prod_{(u,v) \in C} p(u, v)$ revient à minimiser $\sum_{(u,v) \in C} -\ln(p(u, v))$: on peut alors appliquer un algorithme de plus courts chemins classique.
   2ème solution: modifier un algorithme de plus court chemin. Par exemple, Dijkstra deviendrait:

Algorithme de Dijkstra pour chemins les plus probables

Initialement: `next` contient tous les sommets
`proba.(r) <- 1` et `proba.(v) <- 0`, $\forall$ `v` $\neq$ `r`
Tant que `next` $\neq \emptyset$:
    Extraire `u` de `next` tel que `proba.(u)` soit maximum
    Pour tout voisin `v` de `u`:
        `proba.(v) <- max proba.(v) (proba.(u) * p(u, v))`

   On peut refaire la preuve du cours pour voir que cette version de Dijkstra ne marcherait pas pour des « probabilités » > 1. On peut aussi modifier Floyd-Warshall (valable tant qu'il n'y a pas de cycle de probabilité > 1):

Algorithme de Floyd-Warshall pour chemins les plus probables

Initialiser `d.(u).(v)` $\leftarrow p(u, v)$ si $(u, v) \in \vec{E}$, 0 sinon.

Pour $k = 0$ à $|V| - 1$:
    Pour tout sommet u:
        Pour tout sommet v:
            `d.(u).(v)` $\leftarrow$ `max d.(u).(v) (d.(u).(k) * d.(k).(v))`

5. Comment pourrait-on déterminer le poids minimum d'un cycle dans un graphe orienté? En quelle complexité?
   ▶ On peut calculer la matrice `d` des distances (`d.(u).(v)` est la distance du sommet `u` au sommet `v`) puis on calcule la valeur minimum de `d.(u).(v) + d.(v).(u)`, pour tout `u`, `v`.

6. On suppose donné un graphe pondéré où un sommet est une ville et un arc est une route dont le poids est sa longueur. On part d'une ville s avec une voiture consommant 5L/100km et avec un réservoir rempli de 50L d'essence. On suppose connu le prix de l'essence dans chaque ville. Comment trouver un chemin de coût minimum jusqu'à une autre ville t? Quelle serait la complexité?
   ▶ Problème: une configuration dépend à la fois de la ville et de la quantité d'essence restante. On peut construire le graphe dont les sommets sont les couples (ville, essence), où essence est entre 0 et 50 (litres). On rajoute alors des arcs de (ville, essence) vers (ville, essence + 1) de poids le prix d'un litre d'essence dans la ville (ce qui revient à prendre de

l'essence). On ajoute aussi des arcs de (ville1, essence) vers (ville2, essence - consommation × distance de ville1 à ville2) de poids 0, si l'essence reste ≥ 0. Il suffit alors de trouver un plus court chemin de $(s, 50)$ vers un sommet (t, e), avec e quelconque.

7. Soit $\vec{G}$ un graphe sans arc de poids négatif et s, t deux sommets. Montrer que l'on peut trouver tous les arcs utilisés par au moins un plus court chemin de s à t, en utilisant 2 fois l'algorithme de Dijkstra.
   ▶ Soit (u, v) un arc. S'il existe un plus court chemin C de s à t passant par (u, v) alors, comme le sous-chemin de C de s à u est un plus court chemin et le sous-chemin de v à t est un plus court chemin, $d(s, t) = d(s, u) + w(u, v) + d(v, t)$. Réciproquement, si $d(s, t) = d(s, u) + w(u, v) + d(v, t)$ alors (u, v) est sur un plus court chemin de s à t (celui constitué d'un plus court chemin de s à u, puis (u, v), puis un plus court chemin de v à t). On peut donc calculer $d(s, u)$, $\forall u \in V$, avec l'algorithme de Dijkstra puis $d(v, t)$, $\forall v \in V$. Pour savoir si (u, v) est sur un plus court chemin, il suffit alors de tester si $d(s, t) = d(s, u) + w(u, v) + d(v, t)$.

Exercice 3. Plus courts chemins dans un graphe sans cycle

On veut trouver les distances d'un sommet à tous les autres dans un graphe orienté sans cycle $\vec{G} = (V, \vec{E})$ pondéré par w. On pourra utiliser la fonction `tri_topo : int list array -> int list` du TD 2, renvoyant une liste des sommets d'un graphe sans cycle « compatible » avec les arcs ($(u, v) \in \vec{E} \Longrightarrow u$ est avant v dans la liste).

1. Expliquer comment trouver les distances d'un sommet r à tous les autres dans $\vec{G}$ représenté par liste d'adjacence, en complexité $O(|\vec{E}| + |V|)$.
   ▶ Soit $v_1, ..., v_n$ les sommets de $\vec{G}$ dans un ordre topologique. Alors $d(r, r) = 0$ et $d(r, v_k) = \min_{(v_i, v_k) \in \vec{E}} d(r, v_i) + w(v_i, v_k)$ (si $v_k \neq r$ et il n'y a pas d'arc entrant dans v_k on pose $d(r, v_k) = \infty$). On peut donc calculer $d(r, v_k)$ par k croissant.

2. Écrire une fonction correspondant à la question précédente.

```
let topo_bellman g r =
  let n = vect_length g in
  let d = make_vect n max_int in
  d.(r) <- 0;
  do_list (fun u -> do_list (fun v -> d.(v) <- min d.(v) (sum d.(u) (w u v))) g.(u)) (tri_topo g);;
```

3. Comment modifier l'algorithme précédent pour obtenir des chemins de poids **maximum**? Une modification similaire fonctionnerait-elle avec l'algorithme de Dijkstra?
   ▶ On a alors $d(r, r) = 0$ et $d(r, v_k) = \max_{(v_i, v_k) \in \vec{E}} d(r, v_i) + w(v_i, v_k)$ (si $v_k \neq r$ et il n'y a pas d'arc entrant dans v_k on pose $d(r, v_k) = -\infty$). On peut donc calculer $d(r, v_k)$ par k croissant.

Exercice 4. Algorithme de Johnson

Soit $\vec{G} = (V, \vec{E})$ un graphe orienté pondéré par $w : \vec{E} \longrightarrow \mathbb{R}$ (des poids peuvent être négatifs). On a vu en cours l'algorithme de Floyd-Warshall pour trouver les plus courts chemins de n'importe quel sommet u à n'importe quel autre v. L'algorithme de Johnson résout le même problème, mais possiblement avec une meilleure complexité. L'idée de l'algorithme de Johnson est de modifier les poids de $\vec{G}$ pour les rendre positifs, sans modifier les plus courts chemins. On peut alors appliquer $|V|$ fois l'algorithme de Dijkstra (une fois depuis chaque sommet) pour obtenir tous les plus courts chemins.

1. Soit $h : V \longrightarrow \mathbb{R}$. On définit $w_h : (u, v) \longmapsto w(u, v) + h(u) - h(v)$. Montrer que, dans $\vec{G}$, les plus courts chemins pour w et w_h sont les mêmes et qu'il existe un cycle de poids négatif pour w ssi il en existe un pour w_h.
   ▶ Soit C un chemin de u à v. Notons $u = v_1, v_2, ..., v_k = v$ les sommets utilisés par C. Alors $w_h(C) = \sum_{(v_i, v_{i+1})} w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1}) = w(C) + h(u) - h(v)$ (somme télescopique). $h(u) - h(v)$ ne dépend pas du chemin C, seulement des extrémités. Donc un chemin C de poids minimum de u à v est obtenu quand $w(C)$ est minimum, ce qui montre le résultat.

2. Trouver $h : V \longrightarrow \mathbb{R}$ telle que $w_h \geq 0$. On pourra supposer dans un premier temps que tous les sommets de $\vec{G}$ sont atteignables depuis un sommet r.
   ▶ Soit $h(v) = d(r, v)$. Si $(u, v) \in \vec{E}$, il existe un chemin de r à u de poids $d(r, u)$ (par définition de $d(r, u)$ auquel on peut ajouter (u, v) pour obtenir un chemin de r à v. Comme $d(r, v)$ est le poids d'un plus court chemin de r à v, $d(r, v) \leq d(r, u) + w(u, v)$, d'où $w_h \geq 0$.

3. En utilisant plusieurs fois l'algorithme de Dijkstra, écrire une fonction `johnson` renvoyant la matrice des distances entre toute paire de sommets, dans $\vec{G}$ pondéré par w.
   ▶ On peut calculer h avec l'algorithme de Bellman-Ford

```
let johnson g w =
  let d0 = bellman g w 0 in
  let wh u v = d0.(u) + w u v - d0.(v) in
  let d = make_vect n [||] in
  for u = 0 to n - 1 do
    d.(u) <- dijkstra g wh u
  done; d;;
```

4. Comparer la complexité de `johnson` avec celle de Floyd-Warshall