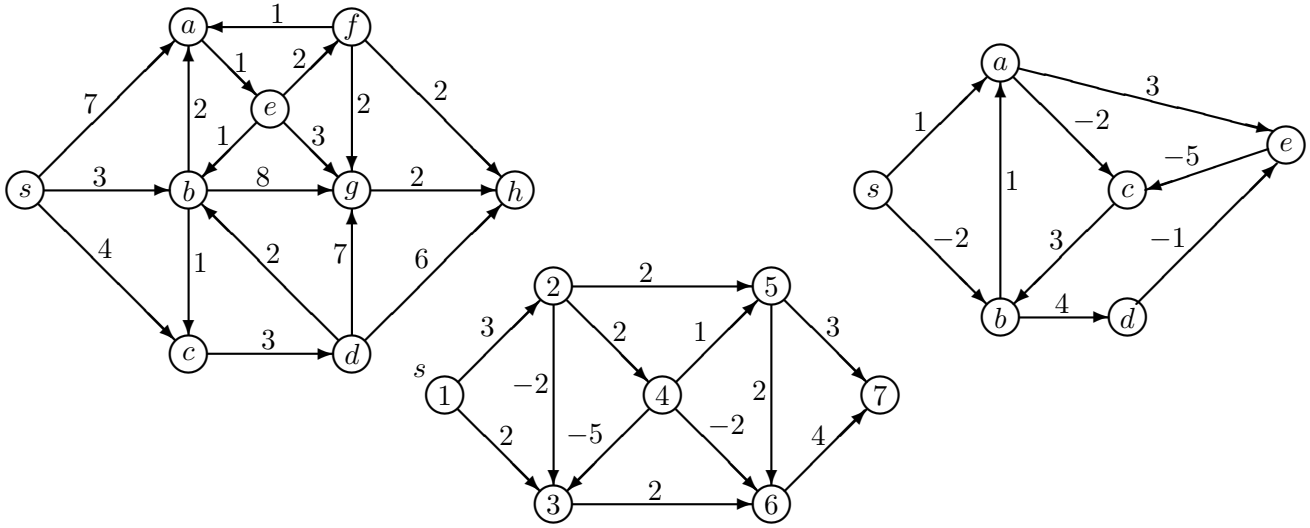
Feuille de TD N^o 4 : Plus courts chemins

1. Utilisez les algorithmes du cours pour déterminer, pour chacun des graphes ci-dessous, une arborescence de plus courts chemins du sommet s à tous les autres.



- *Le premier graphe n'a que des poids positifs : on utilise Dijkstra.*

Sommets visités	s	a	b	c	d	e	f	g	h
$\{s\}$	*0*	*7*	***3***	*4*	∞	∞	∞	∞	∞
$\{s,b\}$	*0*	*5*	*3*	***4***	∞	∞	∞	*11*	∞
$\{s,b,c\}$	*0*	***5***	*3*	*4*	*7*	∞	∞	*11*	∞
$\{s,b,c,a\}$	*0*	*5*	*3*	*4*	*7*	***6***	∞	*11*	∞
$\{s,b,c,a,e\}$	*0*	*5*	*3*	*4*	***7***	*6*	*8*	*9*	∞
$\{s,b,c,a,e,d\}$	*0*	*5*	*3*	*4*	*7*	*6*	***8***	*9*	*13*
$\{s,b,c,a,e,d,f\}$	*0*	*5*	*3*	*4*	*7*	*6*	*8*	***9***	*10*
$\{s,b,c,a,e,d,f,g\}$	*0*	*5*	*3*	*4*	*7*	*6*	*8*	*9*	***10***
Distance finale :	*0*	*5*	*3*	*4*	*7*	*6*	*8*	*9*	*10*

Sommet	s	a	b	c	d	e	f	g	h
Père	-	b	s	s	c	a	e	e	f

Pour le deuxième graphe, aux poids quelconques, on utilise Bellman-Ford.

Étape	s	a	b	c	d	e
0	*0*	∞	∞	∞	∞	∞
1	*0*	*1*	*-2*	∞	∞	∞
2	*0*	*-1*	*-2*	*-1*	*2*	*4*
3	*0*	*-1*	*-2*	*-3*	*2*	*1*
4	*0*	*-1*	*-2*	*-4*	*2*	*1*
5	*0*	*-1*	*-2*	*-4*	*2*	*1*
6	*0*	*-1*	*-2*	*-4*	*2*	*1*

Comme il n'y a pas de changement entre la ligne $n-1$ et la ligne n, on sait que le graphe n'a aucun cycle de poids négatif. L'arbre des plus courts chemins est donné ainsi :

Sommet	s	a	b	c	d	e
Père	-	b	s	e	b	d

Le troisième graphe est acyclique, on peut donc utiliser l'algorithme linéaire en utilisant le tri topologique suivant : 1, 2, 4, 3, 5, 6, 7.

Sommet	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Distance à 1	*0*	*3*	*-1*	*4*	*5*	*5*	*8*
Père	-	*1*	*4*	*2*	*2*	*3*	*5*

2. Dans un pays où la sécurité des chemins n'est pas assurée, on doit aller d'une ville X à une ville Y. Le réseau routier est donné par un ensemble de villes et un ensemble de tronçons de routes joignant ces villes. Pour chaque tronçon t de route, on connait la probabilité P_t de se faire dépouiller sur le tronçon. Comment trouver le chemin de X à Y qui minimise la probabilité de se faire dépouiller ?
   - *La probabilité de se faire détrousser en passant sur un chemin parcourant les portions $x_1, \ldots, x_k$ du réseau routier est le complémentaire de la probabilité de rester indemne sur toutes ses parties ; or, comme ces événements sont indépendants, et que la probabilité de rester indemne sur x_i est $1 - p_{x_i}$, cette probabilité globale est de $(1 - p_{x_1}) \times \ldots \times (1 - p_{x_k})$.*
     On veut donc minimiser un nombre de la forme $1 - (1 - p_{x_1}) \times \ldots \times (1 - p_{x_k})$, c'est-à-dire maximiser un nombre de la forme $(1 - p_{x_1}) \times \ldots \times (1 - p_{x_k})$.
     On peut pour cela adapter un algorithme du cours en remplaçant min *par* max *et les sommes par des produits. L'équivalent d'un cycle absorbant serait un cycle sur lequel le produit des $1 - p_x$ est supérieur à 1 (puisqu'on remplace les sommes par des produits), ce qui n'existe pas puisque tous les $1 - p_x$ sont dans $[0, 1]$.*

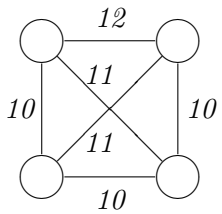
     Plus simplement, comme la fonction log *est strictement croissante et que $\log((1 - p_{x_1}) \times \ldots \times (1 - p_{x_k})) = \log(1 - p_{x_1}) + \ldots + \log(1 - p_{x_k})$, maximiser $(1 - p_{x_1}) \times \ldots \times (1 - p_{x_k})$ revient à minimiser la somme des $-\log(1 - p_x)$ des arêtes d'un chemin de X vers Y, et comme ces valeurs sont toutes positives, on peut utiliser directement Dijkstra sans le modifier pour trouver un chemin optimal.*

3. Donner un algorithme qui prend en entrée un graphe $G = (S, A)$ avec des coûts $w(e)$ éventuellement négatifs sur les arcs, détecte la présence éventuelle d'un cycle absorbant et donne le cas échéant un tel cycle.
   - *On applique l'algorithme de Bellman-Ford en enregistrant le prédécesseur de chaque sommet sur le plus court chemin y menant. Le graphe contient un cycle strictement négatif accessible depuis la source ssi il y a une amélioration lors de la $n^{ième}$ itération, et dans ce cas le cycle est dans la chaîne des prédécesseurs d'un sommet dont la distance à la source diminue lors de cette étape.*
     On remonte alors cette chaîne en marquant un par un les sommets visités jusqu'à tomber sur un sommet déjà visité : on obtient ainsi un cycle contenant ce sommet.
     On note qu'il pourrait y avoir un cycle absorbant non détecté si aucun de ses sommets n'est accessible depuis la source ; dans ce cas on peut travailler comme dans un parcours en largeur ou en profondeur, en recommençant l'algorithme en le lançant d'un sommet pas encore visité (i.e à distance infinie de la source) et en ne s'intéressant qu'aux sommets pas encore visités. On n'empire alors pas la complexité puisque chaue partie du graphe n'est considérée que dans une seule exécution de Bellman-Ford.

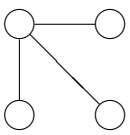
4. Si tous les arcs ont des poids différents, l'arbre des plus courts chemins à partir de s est-il unique ?
   - *Non : considérons le triangle dont un côté est de poids 1, un autre de poids 2 et le troisième de poids 3. Si on part du sommet s situé entre les arêtes 1 et 3 et qu'on cherche à atteindre u situé entre 2 et 3, il y a deux plus courts chemins, tous deux de longueur 3.*

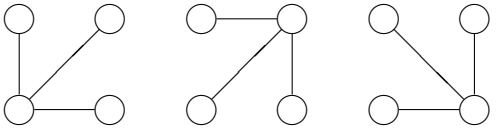
   (maison) Parmi les n (ou plus) arbres de plus courts chemins, au moins l'un d'eux est-il arbre couvrant minimal ?

   - *Non : pour le graphe suivant,*

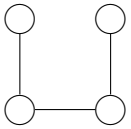


les arbres de plus courts chemins sont





et l'arbre couvrant est



5. Donner un algorithme linéaire qui prend en entrée un graphe orienté $G = (S, A)$, pondéré par $w : E \to \mathbb{R}$, un sommet $s \in S$ et un tableau $T[S]$ de réels et qui teste si pour tout $v \in S$, $T[v]$ est la distance de s à v.

- *Il faut et il suffit de vérifier les choses suivantes :*
  — $T[s] = 0$
  — $\forall v \in S - \{s\}$, $T[v] = min_{\{u|u \to v\}}\{T[u] + w(u \to v)\}$
  — $\forall v \in S$, *si* v *n'est pas accessible depuis* s, *alors* $T[t] = \infty$
  — *les sommets accessibles depuis* s *sont encore accessibles depuis* s *dans le graphe* G' *ou l'on ne garde que les arêtes* $u \to v$ *vérifiant* $T[v] + w(u \to v)$

  Tout cela est bien vérifiable en temps linéaire : les deux premières conditions se vérifient sommet par sommet et arête par arête, en marquant les arêtes qui vérifient le min dans la deuxième condition. Puis on effectue un parcours en partant de s *pour identifier les sommets accessibles, afin de vérifier la troisième conditions, et enfin un autre parcours en ne passant que par les arêtes marquées à la première étape.*
  Montrons maintenant que ces conditions sont nécessaires et suffisantes.

  D'abord, il est évident que si T *contient les distances à* s *de chaque sommet, alors* T *respecte les différentes propriétés.*

  Supposons maintenant que T *respecte ces propriétés. Notons* $d(v)$ *la distance de* s *à* v *pour tout* $v \in S$.
  On va montrer par des récurrences sur des valeurs définies différemment que $T[v] \leqslant d(v)$ *pour tout* v *et que* $T[v] \geqslant d(v)$ *pour tout* v.

  Lemme 1 : $\forall v \in S$, $T[v] \leqslant d(v)$.
  Pour v *non-accessible depuis* s, *c'est évident car* $d(v) = \infty$.
  Soit v *un sommet accessible depuis* s, *i.e* $d(v) < \infty$, *et notons* l *la longueur en nombre d'arête d'un plus court chemin* C *entre* s *et* v. *On va montrer par récurrence sur* l *que le lemme est vrai.*
  Si $l = 0$, *alors* $v = s$ *et donc* $T[v] = 0 = d(v)$.
  Si $l > 0$, *supposons le lemme vrai pour* $l - 1$. *Considérons la dernière arête du plus court chemin* C *de* s *à* v *de longueur* l *; notons-la* $u \to v$. *Alors le chemin* C *auquel on enlève sa dernière arête est un plus court chemin vers* u *(sinon* C *ne serait pas un plus court chemin vers* v*) et prend* $l - 1$ *arête, donc par hypothèse de récurrence,* $T[u] \leqslant d(u)$.
  Le lemme 1 est donc bien vérifié pour tout sommet de G.

  Lemme 2 : $\forall v \in S$, $T[v] \geqslant d(v)$.
  Si v *est non-accessible depuis* s, *alors par la troisième condition,* $T[v] = \infty$ *donc le lemme est vérifié.*
  Soit maintenant v *un sommet accessible dans* G *à partir de* s, *donc également dans* G' *par la quatrième condition. Notons* l *la longueur en arêtes du plus court chemin entre* s *et* V *dans* G'.
  Si $l = 0$, *alors* $v = s$ *et donc* $T[v] = 0 = d(v)$.
  Si $l > 0$, *supposons le lemme vrai pour* $l - 1$. *Considérons la dernière arête du plus court chemin* C' *de* s *à* v *dans* G' *de longueur* $l - 1$ *; notons-la* $u \to v$. *Alors le chemin* C' *auquel on enlève sa dernière arête est un plus court chemin dans* G' *de* s *vers* u *et prend* $l - 1$ *arêtes, donc par hypothèse de récurrence,* $T[u] \geqslant d(u)$.
  Or comme $u \to v$ *est dans* G', *on sait que cette arête vérifie* $T[v] = T[u] + w(u \to v)$. *De plus, on sait que comme tout chemin entre* s *et* u *auquel on ajoute* $u \to v$ *est un chemin de* s *vers* v, $d(v) \leqslant d(u) + w(u \to v)$. *On en déduit qu'on a bien* $T[v] = T[u] + w(u \to v) \geqslant d(u) + w(u \to v) \geqslant d(v)$.
  Ainsi le lemme 2 est bien vérifié pour tout sommet de G.

  On a donc bien prouvé que pour tout $v \in S$, $T[v] = d(v)$.

  Remarque : *la quatrième condition est bien strictement nécessaire à moins qu'on sache que le graphe* G *ne contient aucun cycle de poids nul. Exo en rab : Modifier cet algorithme pour construire une arborescence des plus-courts chemins issus de* s, *toujours ne temps linéaire.*

6. Proposer un algorithme qui ramène le calcul des plus courts chemins depuis une source pour les graphes à valuation dans $\mathbb{N}^*$, à un parcours en largeur. Cet algorithme est-il compétitif face à l'algorithme de Dijkstra ?
   - *En utilisant le fait que tous les poids sont positifs et multiples de 1, on peut transformer le graphe de façon à ce qu'un parcours en largeur suffise à déterminer les distances. Pour cela, on modifie les arêtes ainsi : si l'arête* uv *est de poids* k, *on ajoute* $k - 1$ *sommets entre* u *et* v, *placés en ligne. Ainsi, la distance entre deux sommets est figurée par* k *arêtes de poids 1 plutôt qu'une arête de poids* k.

On peut alors vérifier aisément qu'un plus court chemin dans ce graphe modifié est de même longueur qu'un plus court chemin dans le graphe original, puisque si il passe par une des $w(uv)$ arêtes de poids 1 entre u et v, il passe par toutes. Il y a donc équivalence entre les plus courts chemins dans le graphe original G et le nouveau graphe G'.
La taille de ce nouveau graphe est est $\theta(|S|+|A|+\sum_{a\in A} w(a))$, et le parcours qui permet de déterminer les plus courts chemins dans G' est linéaire en cette taille.
Si les coûts des arcs peuvent être arbitrairement grands, cette technique ne sera donc pas compétitive. En revanche, si les poids sont petits, elle peut être rentable. Si on prend la version de l'algorithme de Dijkstra en $\theta(m \log m$, on obtient que c'est intéressant dans le cas où le coût moyen des arêtes est $o(\log m)$.
Remarque : *On utilise ici la double propriété de $\mathbb{N}$: les nombres sont positifs et tous multiples de 1. On ne peut appliquer cette technique ni dans $\mathbb{Z}$ ni dans $\mathbb{R}^+$.*

7. Soit le graphe dont la matrice d'adjacence est :
(rappel : $A[i, j]$ représente le poids de l'arête de i vers j, et vaut infini si elle n'existe pas) :

$$\begin{bmatrix} 3 & 1 & \infty & 0 & \infty \\ \infty & \infty & 2 & 1 & 3 \\ 4 & \infty & 4 & 1 & 3 \\ 3 & \infty & 1 & 6 & 1 \\ 1 & 4 & \infty & 0 & \infty \end{bmatrix}$$

Rappeler l'algorithme de Floyd-Roy-Warshall et l'utiliser pour calculer la matrice des PCC de tout sommet à tout sommet. Observer comment le plus court chemin de 3 à 2 a été trouvé.

- *Pour le rappel de l'algorithme, se reporter au cours.*
Le graphe étudié a 5 sommets, on va donc calculer 6 matrices qui correspondent à la matrice de départ ($k = 0$) et aux étapes $k = 1$ à 5 de l'algorithme.
Le matrice de départ correspond à la matrice donnée dans l'énoncé, sauf qu'on remplace tous les coefficients diagonaux par des 0, puisqu'on n'a pas besoin d'arêtes pour savoir que la distance d'un sommet avec lui-même est nulle.

Matrice de départ :
$$\begin{bmatrix} 0 & 1 & \infty & 0 & \infty \\ \infty & 0 & 2 & 1 & 3 \\ 4 & \infty & 0 & 1 & 3 \\ 3 & \infty & 1 & 0 & 1 \\ 1 & 4 & \infty & 0 & 0 \end{bmatrix}$$

Étape 1 :
$$\begin{bmatrix} 0 & 1 & \infty & 0 & \infty \\ \infty & 0 & 2 & 1 & 3 \\ 4 & 5 & 0 & 1 & 3 \\ 3 & 4 & 1 & 0 & 1 \\ 1 & 2 & \infty & 0 & 0 \end{bmatrix}$$

Étape 2 :
$$\begin{bmatrix} 0 & 1 & 3 & 0 & 4 \\ \infty & 0 & 2 & 1 & 3 \\ 4 & 5 & 0 & 1 & 3 \\ 3 & 4 & 1 & 0 & 1 \\ 1 & 2 & 4 & 0 & 0 \end{bmatrix}$$

Étape 3 :
$$\begin{bmatrix} 0 & 1 & 3 & 0 & 4 \\ 6 & 0 & 2 & 1 & 3 \\ 4 & 5 & 0 & 1 & 3 \\ 3 & 4 & 1 & 0 & 1 \\ 1 & 2 & 4 & 0 & 0 \end{bmatrix}$$

Étape 4 :
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 4 & 0 & 2 & 1 & 2 \\ 4 & 5 & 0 & 1 & 2 \\ 3 & 4 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 \end{bmatrix}$$

Étape 5 : $\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 3 & 0 & 2 & 1 & 2 \\ 3 & 4 & 0 & 1 & 2 \\ 2 & 3 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 \end{bmatrix}$

Le plus court chemin de 3 à 2 est de longueur 4, d'après cet algorithme. Il est mis à jour à l'étape 5 en additionnant 2 et 2, donc il est de la forme $3 \to_2 5 \to_2 2$.

Or $3 \to 5$ *est mis à jour à l'étape 4 en additionnant 1 et 1, donc le chemin est de la forme* $3 \to_1 4 \to_1 5 \to_2 2$*, et comme ces 1 dont des coefficients de la matrice originale, les deux premières flèches représentent des arêtes.*

De même, $5 \to 2$ *est mis à jour à l'étape 1 en additionnant 1 et 1, qui sont des coefficients originaux, donc le chemin entre 5 et 2 est* $5 \to_1 1 \to_1 2$ *et ces flèches sont des arêtes.*

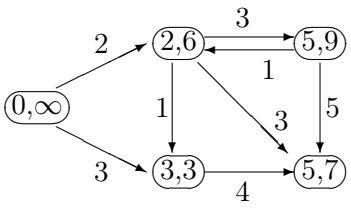
Ainsi, le plus court chemin entre 3 et 2 est le suivant :

$$3 \to_1 4 \to_1 5 \to_1 1 \to_1 2$$

8. **Deuxième distance** (exam 2015-16)

Soit G un graphe valué et s un sommet. Pour tout sommet t, on note $d[t]$ la distance de s à t, i.e. la longueur du plus court chemin de s à t, et $d2[t]$ la deuxième distance de s à t, i.e. la longueur du deuxième plus court chemin de s à t.

Exemple, dans le graphe ci-dessous, s est le sommet à gauche, la distance et la deuxième distance sont notées sur les sommets.



(A) Le graphe G est supposé sans cycle absorbant.

Complétez le code ci-dessous (Bellman-Ford) pour calculer les tableaux d et $d2$.

```
pour tout sommet t, d[t] <- infini
d[s] <- 0
pred[s] <- inexistant

faire n fois :
   pour tous les sommets y faire
      pour tout successeur z de y faire
         si d[z] > d[y] + w(y,z)
         alors   d[z] <- d[y] + w(y,z)
                 pred[z] <- y
```

- *Une deuxième distance d_2 entre s et z peut avoir deux types de valeurs : soit l'avant-dernier sommet y du deuxième plus court chemin est le même que celui du plus court chemin, et alors $d_2(z) = d_2(y) + w(y,z)$, soit l'avant-dernier sommet y est différent de celui du plus court chemin, et dans ce cas $d_2(z) = d(y) + w(y,z)$.*

  On met donc à jour les deux valeurs d et d_2 à chaque étape en testant ces deux possibilités.

```
d <- tableau(n,infini);  /* on initialise d par n valeurs toutes à l'infini */
d2 <- tableau(n,infini);
pred <- tableau(n,indéfini);  /* tableau des prédécesseurs pour le pcc */
d[s] <- 0;
pour k = 1 à n faire
    pour tout sommet y faire
        pour tout z successeur de y faire
```

```
            si d[z] > d[y] + w(y,z)
            alors d[z] <- d[y] + w(y,z);
                  pred[z] <- y;
            fin si;
            si y = pred[z] et d2[z] > d2[y] + w(y,z)
            alors d2[z] <- d2[y] + w(y,z);
            fin si;
        fin pour;
    fin pour;
fin pour;
renvoyer d,d2;
```

(B) Les valuations de G sont supposées positives.
Complétez le code ci-dessous (Dijkstra) pour calculer les tableaux d et $d2$.

```
pour tout sommet t, d[t] <- infini
d[s] <- 0
E <- S

faire n fois :
   soit y dans E tq d[y] minimise { d[w] | w dans E }
   E <- E - { y }
   pour tout successeur z de y faire
      si d[z] > d[y] + w(y,z)
      alors   d[z] <- d[y] + w(y,z)
```

- *Cette fois-ci, on va effectuer deux passages de Dijkstra successifs sur le graphe. Le premier calcule les distance classiques, et le second les deuxièmes distances, en initialisant la valeur de $d_2(z)$ au min des $d(y)+w(y,z)$ pour y un prédécesseur de z qui n'est pas son prédécesseur sur le plus court chemin entre s et z.*
  L'algorithme fonctionnera car un deuxième plus court chemin a une forme très spécifique : si T est l'arbre des plus courts chemins trouvé lors du premier passage de Dijkstra, alors :
  — *soit $d_2(z) = d_2(y) + w(y,z)$ où y est le prédécesseur de z dans T, donc le deuxième pcc de s à z est le deuxième pcc de s à y plus l'arête (y,z) qui est dans T ;*
  — *soit $d_2(z) = d(y) + w(y,z)$ où y n'est pas le prédécesseur de z dans T, et donc le deuxième pcc de s à z est le pcc de s à y (inclu dans T) plus l'arête (y,z), qui n'est pas dans T.*
  Dans tous les cas, un deuxième pcc contient toujours exactement une arête qui n'est pas dans T : si il n'en contenait pas il serait un pcc, mais si il en contient une, toutes celles d'avant sont dans T (cas 2) et toutes celles d'après sont aussi dans T (cas 1).
  Donc on peut modifier la preuve habituelle de la correction de Dijkstra en faisant une récurrence sur le nombre d'arêtes après l'arête hors de T dans le deuxième plus court chemin de s à v.

```
d <- tableau(n,infini);
d[s] <- 0;
pred[s] <- (n, indéfini);
E <- S;
pour k = 1 à n faire
    soit y dans E tq d[y] minimise { d[w] | w dans E };
    E <- E - { y };
    pour tout successeur z de y faire
        si d[z] > d[y] + w(y,z)
        alors d[z] <- d[y] + w(y,z);
              pred[z] <- y;
        fin si;
    fin pour;
fin pour;
```

```
d2 <- (n, infini);
pour tout y sommet faire
    pour tout successeur z de y faire
        si y != pred[z] alors d2[z] <- min(d2[z], d[y]+w(y,z));
    fin pour;
fin pour;
E <- S;
pour k = 1 à n faire
    soit y dans E tq d2[y] minimise { d[w] | w dans E };
    E <- E - { y };
    pour tout successeur z de y faire
        si y = pred[z] alors d2[z] <- min(d2[z], d2[y] + w(y,z));
    fin pour;
fin pour;
renvoyer d,d2;
```