L3/Info-MIAGE **2017-2018**

Algorithmique des graphes : examen de décembre

Uniquement support du cours autorisé

Concepteurs : Rumen Andonov, Mikail Demirdelen, Sébastien François, Sébastien Lê Cong

Veuillez écrire chaque partie sur une nouvelle feuille afin de permettre la parallélisation de la correction

*Le barème est donné à titre **indicatif** seulement.*

Exercice 1 (Stage à la mairie (12,5 points + 2 points bonus))
Un étudiant de l'ISTIC effectue son stage à la mairie de Rennes dans un projet qui doit optimiser l'approvisionnement de quatre nouveaux clients dans la ville. Ces quatre clients (notés C_1, C_2, C_3 et C_4) doivent être approvisionnés à partir de quatre dépôts de stocks notés D_1, D_1, D_2, D_4. Les données du projets sont les suivantes.

- La demande hebdomadaire du client C_1 s'élève à 20 unités. Ce client est approvisionné à partir des dépôts D_2, D_3, D_4.
- La demande hebdomadaire du client C_2 s'élève à 25 unités. Ce client est approvisionné à partir du dépôt D_2.
- La demande hebdomadaire du client C_3 s'élève à 15 unités. Ce client est approvisionné à partir des dépôts D_1, D_2, D_4.
- La demande hebdomadaire du client C_4 s'élève à 10 unités. Ce client est approvisionné à partir des dépôts D_2, D_4.

Les capacités des dépôts sont comme suit :

D_1	D_2	D_3	D_4
20	20	10	15

. Il s'agit de satisfaire les demandes des clients en minimisant la quantité du stock circulant sur le réseau.

Cet étudiant est un passionné d'algorithmique des graphes et découvre avec satisfaction que les tâches qui lui sont attribuées sont en parfaite concordance avec les connaissances qu'il a acquises en cours. Votre travail consiste à donner votre propre réponse à chacune de ces tâches. Ces tâches sont indépendantes et peuvent être abordées séparément (sauf la première tâche qui est commune et obligatoire).

1.1 T1 : *Décrire le graphe associé au contexte du projet. (1 point)*

PARTIE 1 : Programmation linéaire (6 points + 2 points bonus)

Pour résoudre le problème d'optimisation l'étudiant envisage d'utiliser un solveur linéaire. Il propose un programme linéaire ayant comme objectif de déplacer le moins d'unités possible tout en satisfaisant la demande des clients et en respectant le stock disponible. Son programme considère les quatre clients, les quatre dépôts et les approvisionnements donnés.

1.2 T2 : *Donner ce programme linéaire. Les variables, l'objectif, ainsi que chaque contrainte seront explicités sans ambiguïté. (2 points)*

L'étudiant s'aperçoit ensuite qu'en cas de changement des données, son code ne sera plus applicable. Il décide alors de créer un code plus universel et écrit le modèle linéaire pour le cas général, capable de résoudre le même problème quel que soit le graphe biparti $G = (D \cup C, E)$. Pour écrire le code d'une manière plus formelle, il faut connaître l'ensemble des clients que chaque dépôt peut approvisionner. Il est donc clair que nous avons besoin de la liste des successeurs pour correctement modéliser notre problème. Nous pouvons construire cette liste, puisque nous avons accès à la liste des prédécesseurs.

1.3 T3 : *Donner le pseudo-code de l'algorithme qui permet de calculer la liste des successeurs à partir de celle des prédécesseurs. En considérant les opérations élémentaires effectuées sur les listes, quelle est sa complexité ? (3 points)*

1.4 T4 : *Donner ensuite le programme linéaire dans le cas général. (1 point)*

PARTIE 2 : Résolution à la main (5,5 points)

Malheureusement, le jour J, le solveur tombe en panne. L'étudiant ne panique pas puisqu'il s'était déjà aperçu qu'il maitrise une deuxième approche de résolution du problème d'approvisionnement. Fondée sur un grand classique de l'algorithmique des graphes, cette approche lui permet d'obtenir une solution à la main. Il suffit d'ajouter au graphe en question deux sommets artificiels et de les connecter convenablement aux sommets existants. Il suppose que les capacités des routes connectant les clients et les dépôts sont suffisamment grandes par rapport aux données du problème.

1.5 T5 : *Quelle est cette approche ? Appliquer cette approche au problème pour montrer que ce dernier n'a pas de solution admissible. Quelle est d'après vous la justification que l'étudiant a donnée, c'est-à-dire la preuve mathématique que le problème de l'approvisionnement optimal est résolu ? Il s'est aussi aperçu que certains clients ne peuvent pas être suffisamment approvisionnés. Quels sont ces clients dans votre simulation ? (3 points)*

Les acteurs principaux du projet se réunissent alors en urgence pour proposer une solution de sortie de crise. Le directeur du réseau routier propose de construire deux nouvelles routes pour améliorer la connectivité des clients démunis. Chacune de ces routes coûtera 100 000 €. Le directeur de la société propriétaire des dépôts de stock s'oppose à cette proposition en donnant un argument pertinent pour convaincre l'audience que le problème n'est pas dû à la connectivité des clients avec les dépôts.

1.6 T6 : *Quel est cet argument d'après vous ? (0,5 point)*

Pour sortir de la crise, le même directeur propose d'augmenter la capacité d'un des dépôts avec 10 unités de stock.

1.7 T7 : *Quel est d'après vous ce dépôt ? Justifier que l'augmentation de sa capacité peut résoudre le problème. (2 points)*

BONUS

Cet élargissement s'effectuant par tranche de 5 unités, et chaque tranche s'élevant à 200 000 €, il demande un montant de 400 000 € à la mairie, ce qui est inacceptable dans le contexte actuel du budget de la mairie.
Après avoir analysé les propositions des deux directeurs, l'étudiant expose sa propre solution qui ne nécessite que 300 000 €. Il donne des arguments pertinents pour convaincre l'audience qui doute fortement de cette proposition inattendue.

1.8 T8 : *Quelle est d'après vous la proposition de l'étudiant ? Donner sa justification (la preuve mathématique qu'elle résout le problème d'approvisionnement des quatre clients). (2 points)*

Exercice 2 (Ulysse revient, et c'est un bien long chemin (7,5 points + 1 point bonus))
Ulysse, errant à travers l'espace intersidéral, pilote son vaisseau de haute technologie, l'Odysseus ; pour explorer une galaxie donnée, partant d'une certaine étoile de départ σ, il souhaite se rendre à une autre étoile prédéterminée φ, et ceci en le moins de temps possible. Grâce à son ordinateur de bord, la durée (exprimée en jours) du trajet direct entre deux étoiles données est connue.

À la suite d'une fausse manœuvre, lui et tout son équipage se retrouvent coincés dans la menaçante galaxie Ω, véritable Triangle des Bermudes de l'espace, riche en trous de ver où plus d'un nerd a disparu après s'y être aventuré ! Un trou de ver entre deux étoiles est comme un trajet direct, à ceci près qu'il fait remonter dans le temps les voyageurs qui l'empruntent.

Qui plus est, **le vaisseau se retrouve bloqué sur le pilotage automatique** : il cherchera donc systématiquement à diminuer la durée du trajet, quitte à se retrouver coincé dans un *trou noir spatio-temporel* (voir Figure 1), c.-à-d. un circuit où on remonte dans le temps sans jamais pouvoir s'arrêter ! Cependant, tout n'est peut-être pas perdu, car la sagacité d'Ulysse peut lui permettre de modifier les données de l'ordinateur de bord.

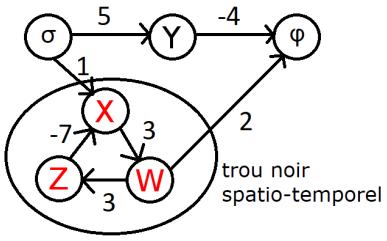


FIGURE 1 – Un **exemple** de situation dangereuse : partant de σ, bien que le vaisseau pourrait passer par Y, le pilote automatique va aller en X pour ensuite remonter le temps à l'infini le long de $X - W - Z - X$, et le vaisseau n'arrivera jamais en φ.
Ici, par exemple : voyager de X vers W prend 3 jours, et on remonte de 7 jours dans le temps en allant de Z vers X.

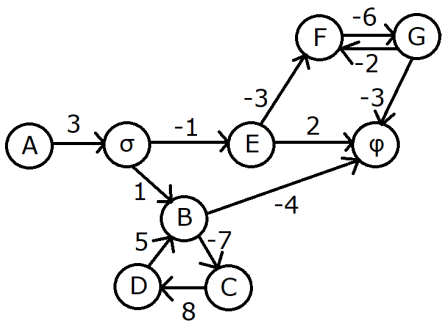


FIGURE 2 – La galaxie Ω, **que nous regardons dans cet exercice**, est représentée par ce graphe G.

Notre but est d'étudier la situation pour voir si Ulysse et les siens vont se sortir de ce mauvais pas ; pour cela, nous allons modéliser le problème par un graphe orienté G où chaque étoile correspond à un sommet, un arc entre deux étoiles indique un trajet direct entre elles et son poids dénote la durée dudit trajet en jours (un poids strictement négatif représentant un retour dans le temps). La Figure 1 est un simple **exemple** illustratif.

Pour ce faire, nous suivrons le plan d'action suivant pour traiter **le graphe G correspondant à la galaxie Ω (voir Figure 2)**, constitué de quatre étapes :

1. *Partitionner* le graphe en zones et en déduire un graphe simplifié de la galaxie (voir **E1**) ;
2. *Détecter* les zones à trou noir (voir **E2**) ;
3. *Élaguer (couper)* le graphe simplifié pour éviter au vaisseau d'aller dans des zones dangereuses (voir **E3**) ;
4. Enfin seulement, essayer de *calculer* un trajet par le biais d'un algorithme (voir **E4**).

PARTIE 3 : Zones (5 points)

Nous allons commencer par cartographier la galaxie en zones, afin de mieux voir quelles étoiles sont dangereuses ou pas. Une *zone* Z sera, par convention, un ensemble d'étoiles telles qu'il existe un chemin de n'importe quelle étoile de Z vers n'importe quelle autre de la même zone Z ; de plus, quand on passe d'une zone Z_1 à une zone Z_2, il ne doit pas être possible de revenir dans la zone Z_1. Enfin, toute étoile appartient à une zone.

2.1 **E1 :** *À quelle notion vue en cours correspondent ces zones ? Comment appelle-t-on le graphe simplifié $\widetilde{G}$ qui représente les déplacements possibles entre zones ? En utilisant un algorithme vu en cours, déterminer ce graphe $\widetilde{G}$ (il* **n'est pas pondéré/valué** *: les arcs ne comportent pas de valeur).* ⚠ *Chaque étape de l'algorithme doit être détaillée. (4 points)*

Nous souhaitons maintenant distinguer les zones comportant un trou noir (dites *zones dangereuses*) des autres zones (dites *zones sûres*).

2.2 **E2 :** *Considérons une zone (un sous-graphe de G composée de n étoiles). Quel algorithme peut-il permettre de détecter si elle comporte un trou noir ou pas, et comment ?*
*Dans votre graphe $\widetilde{G}$, marquer d'une croix la ou les zone(s) dangereuse(s). (**On ne demande pas d'appliquer l'algorithme ici** : vous n'avez pas à justifier pourquoi les zones sont dangereuses ou sûres ; le graphe est suffisamment petit pour le voir à la main.)*
(1 point)

PARTIE 4 : Trajet (2,5 points + 1 point bonus)

Nono le robot informe Ulysse d'une bonne nouvelle : il est possible de supprimer les coordonnées de zones choisies dans la base de données du vaisseau, lui empêchant ainsi de s'y rendre ! Ulysse se rue vers la console et ouvre un terminal Linux pour supprimer certaines données de la mémoire.

2.3 **E3 :** *Quelle zone et donc quelles étoiles Ulysse peut-il légitimement effacer ? En déduire le graphe $G_{\text{sûr}}$: un sous-graphe du* ***graphe initial*** *G privé de ces zones. (0,5 point)*

Il s'agit maintenant de vérifier si l'Odysseus pourra arriver à bon port, et si oui, comment (on rappelle que le vaisseau cherche toujours à minimiser la durée du voyage).

2.4 **E4 :** *Dans le nouveau graphe $G_{\text{sûr}}$* ***(où il y a une valeur sur les arcs)****, quel est algorithme vu en cours le plus efficace que l'on peut légitimement appliquer pour trouver un trajet convenable ? Justifier en citant notamment le critère d'application de l'algorithme retenu et celui des algorithmes concurrents éventuellement écartés s'il y en a. Le graphe étant suffisamment simple,* ***on ne demande pas d'utiliser l'algorithme*** *: trouver à la main une solution du problème, c'est-à-dire un trajet optimal et sa durée en jours. (2 points)*

BONUS

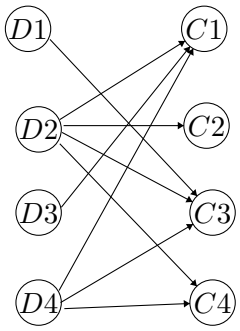
Imaginons que, dans une galaxie représentée par un graphe orienté pondéré quelconque G_0, l'ordinateur de bord de l'Odysseus n'ait pas été reprogrammable et qu'Ulysse n'ait donc pas été autorisé à supprimer des données...

2.5 **E5 :** *Donner une condition (sous forme de phrase) nécessaire et suffisante portant sur les chemins reliant σ à φ pour que le vaisseau puisse quand même arriver automatiquement à destination. (1 point)*

Remarque / prise de distance (!) : *cette dernière question montre que même dans un graphe avec des circuits absorbants, on peut parfois trouver des plus courts chemins depuis un point de départ pour aller à certains sommets. Cet exercice s'est penché plus précisément sur ce cas.*
Remarque bis : *référence culturelle du contexte pour les curieux :* Ulysse 31.

Solutions

Solution 1.1.



Solution 1.2. Notons x_{ij} la quantité du stock circulant du dépôt i au client j, $x_{ij} \geq 0$.

Minimiser z	$=$	x_{13}	$+x_{21}$	$+x_{22}$	$+x_{23}$	$+x_{24}$	$+x_{31}$	$+x_{41}$	$+x_{43}$	$+x_{44}$	
s. c.	$D_1:$	x_{13}									≤ 20
	$D_2:$		x_{21}	$+x_{22}$	$+x_{23}$	$+x_{24}$					≤ 20
	$D_3:$						x_{31}				≤ 10
	$D_4:$							x_{41}	$+x_{43}$	$+x_{44}$	≤ 15
	$C_1:$		x_{21}				$+x_{31}$	$+x_{41}$			≥ 20
	$C_2:$			x_{22}							≥ 25
	$C_3:$	x_{13}			$+x_{23}$				$+x_{43}$		≥ 15
	$C_4:$					$+x_{24}$				$+x_{44}$	≥ 10

Solution 1.3. Soit le graphe biparti $G = (V, E) = (D \cup C, E)$ avec D l'ensemble de dépôts, et C l'ensemble de clients.

Pour chaque sommet v dans D on crée une liste vide de successeurs. (il y a donc une liste vide par sommet). Coût de cette opération : $|V|$.
Puis pour chaque sommet u dans C : on parcourt un a un les éléments v de la liste des prédécesseurs qui lui sont associés. A chaque fois on ajoute u comme successeurs de v en ajoutant u à la liste des successeurs qui lui est associée : `succ(v).add(u).`
(Complexité : $|E|$ (pour chaque sommet on parcourt chaque prédécesseur : on parcourt donc chaque arc du graphe).

```
for v in D do
   succ(v) := []
endfor
for u in C do
  for v in pred(u) do
```

```
     succ(v).add(u)
  endfor
endfor
```

Complexité : $O(|V| + |E|)$.

Version acceptable :

```
for v in D do
   succ(v) := null
endfor
for v in C do
  while(pred(v) ≠ null) do
     u := eject(pred(v))
     inject(v,succ(u))
  endwhile
endfor
```

Solution 1.4. Notons s_i la quantité du stock disponible dans le dépôt D_i, et d_i la quantité de la demande du client C_i.

$$\text{Minimiser} \sum_{(i,j)\in E} x_{(i,j)} \tag{1}$$

$$\forall i \in D : \sum_{j\in\Gamma^+(i)} x_{(i,j)} \leq s_i \tag{2}$$

$$\forall j \in C : \sum_{i\in\Gamma^-(j)} x_{(i,j)} \geq d_j \tag{3}$$

Solution 1.5.
On considère le graphe à la Figure 3. Le sommet source S est connecté à chaque dépôt D_i et l'arc (S, D_i) est muni d'une capacité s_i. Le sommet puits T est connecté à chaque client C_i et l'arc (C_i, T) est muni d'une capacité d_i.
La justification consiste à dire que la valeur du flot maximum (60 unités) est inférieure au total de la demande (70 unités). Dans la solution du graphe à la Figure 3, la demande des clients C_1, C_2 n'est pas satisfaite.
Solution 1.6. La demande totale vaut 70, tandis que le stock total vaut 65. Donc, on ne peut jamais satisfaire la demande.
Solution 1.7. Le dépôt en question est D_2. On voit facilement que l'augmentation de sa capacité de 20 à 30 unités permet de résoudre le problème d'approvisionnement (le graphe n'est pas visualisé, mais on s'attend à que les étudiants le fassent.)
Solution 1.8.
Solution 2.1. Ce sont les composantes fortement connexes ; $\widetilde{G}$ s'appelle le graphe réduit ou métagraphe.
Un seul algorithme vu en cours pour son calcul (algorithme de Kosaraju, mais il en existe d'autres, comme celui de Tarjan ou celui de Gabow ; cf. article « Composante fortement connexe » sur

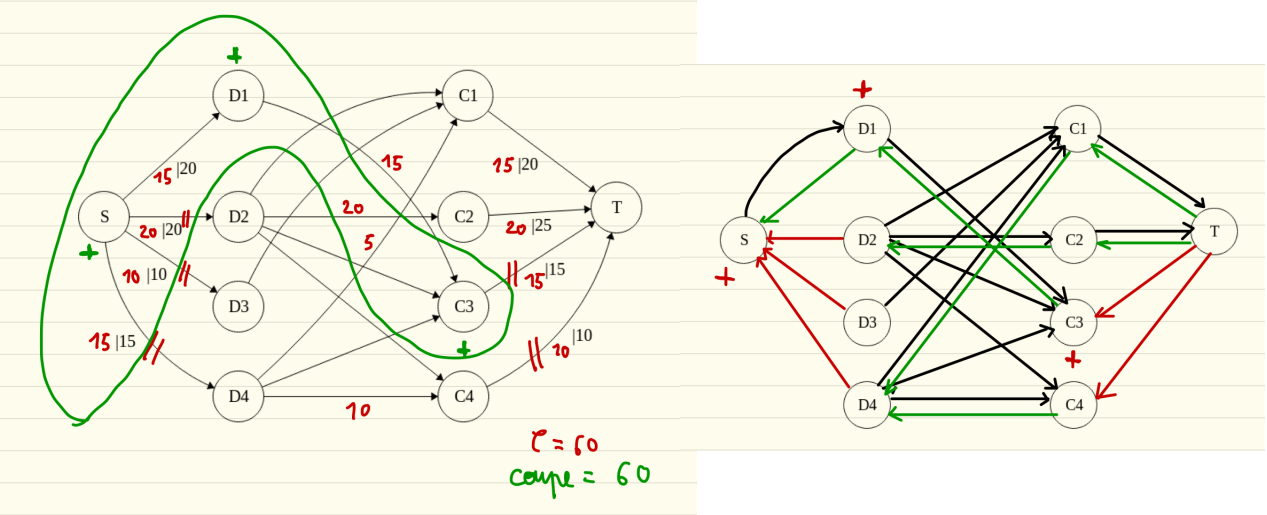
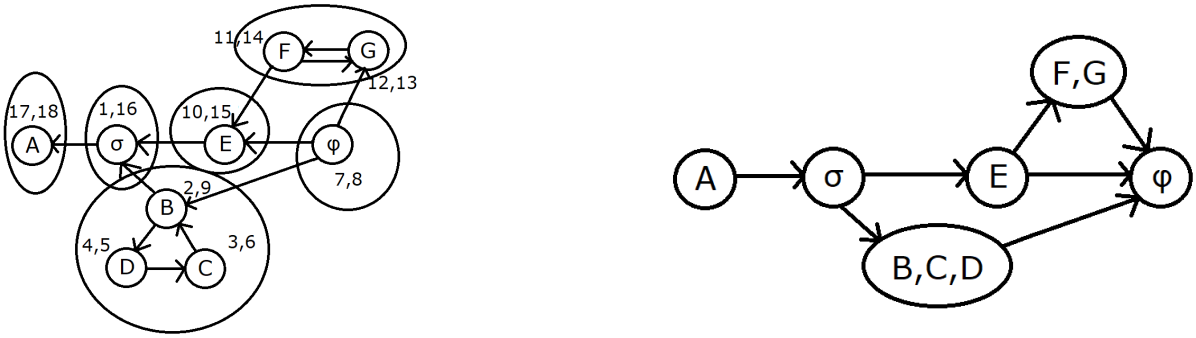


FIGURE 3 – **Gauche :** le graphe après l'ajout des sommets source S et puits T. Les valeurs du flot sont en rouge, la coupe minimale est indiquée en vert. La valeur du flot est égale à la valeur de la coupe — preuve que c'est le flot max. **Droite :** le graphe résiduel qui permet de trouver la coupe minimale.

Wikipédia) : parcours en profondeur avec annotation des sommets par les valeurs *pre* et *post*, puis autre parcours en profondeur mais dans le graphe renversé $\overline{G}$ en priorisant les sommets par valeur *post* décroissante. Ci-dessous, à gauche, un exemple d'annotation (*pre*, *post*) (cette dernière dépendant du premier parcours en profondeur effectué) sur le graphe renversé ; à droite, une représentation possible du graphe réduit $\widetilde{G}$:



Solution 2.2. L'algorithme de Bellman-Ford (initialisé depuis n'importe quel sommet du sous-graphe, puisque tout sommet y est accessible) ; en numérotant 0 l'initialisation, s'il ne s'est toujours pas stabilisé à la n^e itération, on a détecté un circuit de poids strictement négatif, c'est-à-dire dans ce contexte un trou noir.
Le graphe réduit $\widetilde{G}$ marqué peut être représenté comme suit :

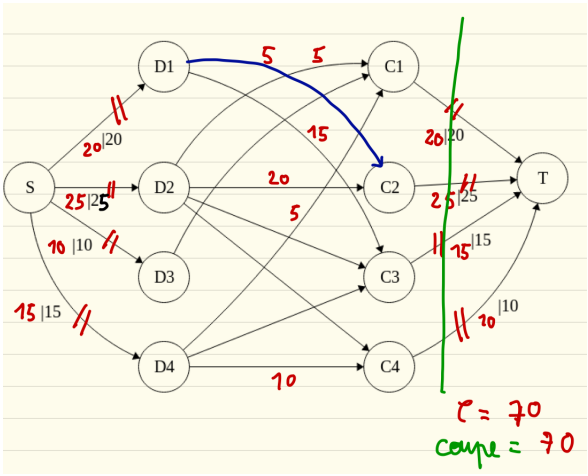
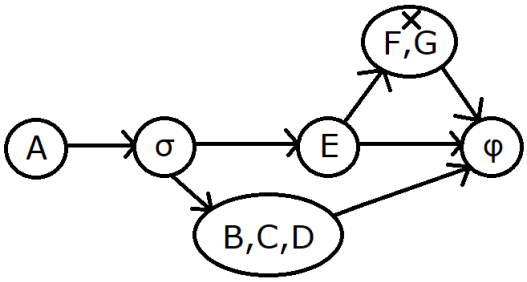
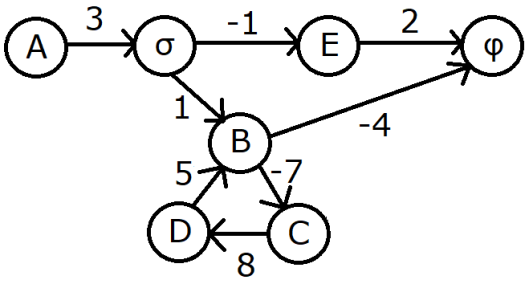


FIGURE 4 – On a ajouté un nouvel arc (D_1, C_2) (coût = 100 000 €). On a aussi augmenté la capacité du dépôt D_2 de 20 à 25 unités (coût = 200 000 €). Ceci a permis d'obtenir un flot de volume 70. On a aussi visualisé une coupe de la même capacité — preuve que c'est le flot max.



Solution 2.3. On peut supprimer toutes les zones dangereuses ; ici, seule la zone $\{F; G\}$ est dangereuse. Cela correspond à retirer les étoiles (sommets) F et G du graphe initial (et les arcs qui vont avec).
Le graphe $G_{\text{sûr}}$ élagué peut être représenté comme suit :



Solution 2.4. Il y a des poids négatifs, donc pas l'algorithme de Dijkstra ; il reste de plus un circuit, donc pas possible de trier ou étager le graphe en vue de programmation dynamique. En

revanche, il n'y a pas/plus de circuit de poids strictement négatif : en conséquence, l'algorithme de Bellman-Ford (initialisé à σ) peut être utilisé.
Ici, le chemin $\sigma \to B \to \varphi$ convient (c'est d'ailleurs le seul plus court chemin), et sa durée est de -3 jours (autrement dit, on remonte de 3 jours dans le temps).

Solution 2.5. D'une part, il existe un chemin joignant σ à φ, et d'autre part, aucun chemin reliant σ à φ ne passe par une zone dangereuse.