

TD2 - Plus courts chemins

GRAPHES ET PROGRAMMATION DYNAMIQUE

M1 Informatique - Semestre 1 - Année 2022-2023

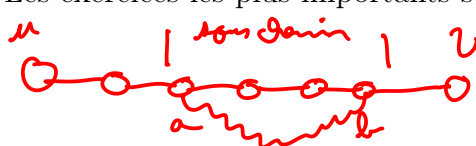
UNIVERSITÉ CÔTE D'AZUR

Christophe Crespelle

christophe.crespelle@univ-cotedazur.fr

Le sujet sera traité sur une séance d'1h30. Il est recommandé que vous utilisiez les exercices que vous n'avez pas eu le temps de faire en séance comme entraînement en vue de l'examen final. Les exercices les plus importants sont les 1, 2 et 3 : commencez par ceux-là.

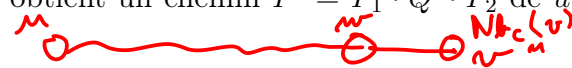
Exercice 1.



$Nbc_u(x)$
Différentes façons de cheminer.

a. Montrez qu'un sous-chemin d'un plus court chemin est aussi un plus court chemin.

Solution. Si un chemin $P = P_1 \cdot Q \cdot P_2$ de u à v contient un chemin Q de a à b qui n'est pas un plus court chemin de a à b . Alors en remplaçant Q par un plus court chemin Q' de a à b dans P , on obtient un chemin $P' = P_1 \cdot Q' \cdot P_2$ de u à v qui est plus court que le chemin P .



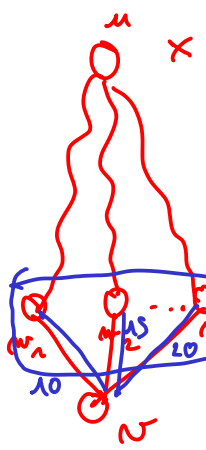
b. Soit $G = (V, E)$ un graphe non orienté et non valu. Soit $u, v \in V$ et soit $w \in V$ tel que wv est la dernière arête d'un plus court chemin de u à v . On note $Nbc_u(v)$ le nombre de plus courts chemins de u à v . Quel est le nombre $Nbc_u(wv)$ de plus courts chemins de u à v qui terminent par l'arête wv ?

Solution. Si wv est la dernière arête d'un plus court chemin de u à v alors la première partie P de ce chemin est un plus court chemin de u à w , d'après la question précédente. Réciproquement, si P' est un plus court chemin de u à w , alors $P' \cdot wv$ est un plus court chemin de u à v , car sa longueur est bien la distance de u à v . Ainsi, les plus courts chemins de u à v terminant par wv sont en même nombre que les plus courts chemins de u à w : $Nbc_u(wv) = Nbc_u(w)$.

c. On note $Prec_u(v)$ l'ensemble des sommets $w \in V$ tels que wv est la dernière arête d'un plus court chemin de u à v . Exprimez $Nbc_u(v)$ en fonctions des $Nbc_u(w)$ pour $w \in Prec_u(v)$.

$$Nbc_u(v) = Nbc_u(w_1) + Nbc_u(w_2) + \dots + Nbc_u(w_n)$$

Solution. On a $Nbc_u(v) = \sum_{w \in Prec_u(v)} Nbc_u(wv) = \sum_{w \in Prec_u(v)} Nbc_u(w)$, d'après la question précédente.



Dans la suite de l'exercice, le graphe G donne en entree est non oriente et non pondere.

d. Adaptez l'algorithme de parcours en largeur depuis u pour qu'en plus de la distance a u , il calcule pour chaque sommet v , le nombre de plus courts chemins de u a v . Quel est la complexite de l'algorithme adapte ?

Solution. Pour compter le nombre de plus courts chemins de u a v , il suffit de determiner l'ensemble $Prec_u(v)$. Cet ensemble et celui des sommets w tels que lors du parcours des voisins de w on trouve v qui est soit blanc, soit gris avec $d[v] = d[w] + 1$. Ainsi, au cours du BFS, lorsqu'on est dans une de ces deux situations, on actualise $Nbc_u(v)$ par $Nbc_u(v) \leftarrow Nbc_u(v) + Nbc_u(w)$ ($Nbc_u(v)$ est initialise a 0 au debut de l'algorithme). Cela ne change pas la complexite de l'algorithme car le surcout de cette operation, test + actualisation, est constant pour chaque traversee d'arete : on a toujours une complexite de $O(n + m)$.

e. Adaptez l'algorithme de parcours en largeur depuis u pour qu'il determine et stocke pour chaque sommet u la liste des sommets dans $Prec_u(v)$. Quelle est la taille de la representation ainsi fournie en sortie de l'algorithme ?

Solution. Si on veut en plus stocker la liste $Prec_u(v)$, il suffit d'ajouter en tete de liste chaque sommet w qui satisfait la condition de la question precedente. La taille de $Prec_u(v)$ est au plus $d^\circ(v)$ et la taille totale de la representation fournie en sortie de l'algorithme, arbre BFS + listes $Prec_u(v)$, devient donc $O(n + m)$, car $\sum_{v \in V} d^\circ(v) = 2m = O(m)$

f. Etant donne la representation fournie en sortie de l'algorithme a la question precedente, comment tirer uniformement aleatoirement un plus court chemin de u a v . Quelle est la complexite d'un tel tirage aleatoire ?

Solution.

On peut proceder comme suit. D'abord, il faut au prealable avoir choisi un ordre $p_1(v), p_2(v), \dots$ sur les sommets de $Prec_u(v)$ pour chaque v , par exemple l'ordre dans lequel ils se presentent dans la liste qui stocke $Prec_u(v)$. Cela induit alors un ordre total sur les plus courts chemins de u a v , en prenant l'ordre lexicographique sur le chemin lu de v a u .

Pour effectuer un tirage de plus court chemin, on tire alors un entier $rand$ uniformement aleatoirement dans l'intervalle $\llbracket 1, Nbc_u(v) \rrbracket$. Ensuite, on determine chaque sommet sur le chemin tire en partant de $x = v$. A la premiere etape, on commence par parcourir les sommets w de $Prec_u(x)$ en sommant leurs $Nbc_u(w)$ dans une variable S . Le prochain sommet sur le plus court chemin tire est le premier sommet w tel que $S \geq rand$. On passe alors a l'etape suivante en actualisant x et $rand$ comme suit : $x \leftarrow w$ et $rand \leftarrow rand - (S - Nbc_u(w))$. On cherche alors le prochain sommet sur le chemin tire parmi $Prec_u(x)$ pour le nouveau x . Le processus se termine lorsque $x = u$.

La complexite d'un tel tirage aleatoire est $O(n)$ car pour tout plus court chemin P de u a v , les ensembles $Prec_u(v)$ pour $v \in P \setminus \{u\}$ sont deux a deux disjoints.

Exercice 2.

La charue avant les boeufs.

Quand vous vous levez le matin, pour vous habiller, vous devez mettre vos chaussettes avant vos chaussures, et vous pouvez mettre votre montre à n'importe quel instant. L'habillage d'un cadre moyen peut globalement se représenter par le graphe (orienté) de contraintes donne sur la figure 1, dans lequel un arc du vetement A vers le vetement B signifie que le vetement A doit obligatoirement etre mis avant de mettre le vetement B.

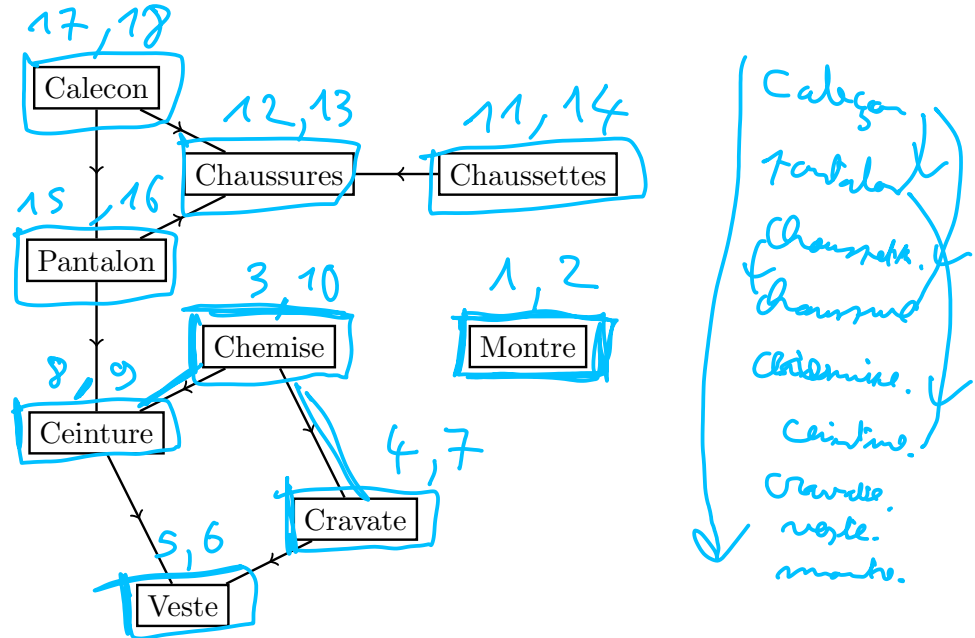


FIGURE 1 – Le graphe des contraintes sur l'ordre d'habillage. Un arc du vetement A vers le vetement B signifie que le vetement A doit obligatoirement etre mis avant de mettre le vetement B.

On se propose de trouver une méthode systematique pour habiller tout le monde, quels que soient ses vetements. C'est a dire, qu'étant donne un graphe de contrainte comme celui de la figure 1, on veut trouver un ordre total sur les vetements à enfiler qui respecte toutes les contraintes du graphe de contrainte.

a. Pour quels graphes de contraintes cela est-il possible? Demontrez le.

Indication. Determinez d'abord pour quels graphes ce n'est pas possible.

Indication. Prouvez que pour les autres c'est toujours possible, par induction sur le nombre de sommets, en choisissant bien le sommet a enlever dans l'induction.

Solution.

S'il y a un circuit dans un graphe oriente, on ne peut pas trouver d'ordre topologique sur ses sommets. On va montrer qu'a l'inverse, si le graphe ne contient pas de circuit, il existe un ordre topologique sur ses sommets.

On va le montrer par recursion sur son nombre de sommets. Pour un graphe a 1 sommet, c'est vrai. Supposons que c'est vrai pour tous les graphes ayant k sommets et montrons que c'est vrai pour un graphe G a $k + 1$ sommets.

Dans ce but, on va d'abord montrer que tout graphe sans circuit possede au moins une source, c'est a dire un sommet qui n'a pas de voisins entrants. On montre la

contraposee : si tous les sommets ont au moins un voisin entrant, on peut choisir iterativement un voisin entrant du sommet courant. Comme il y a un nombre fini de sommet, apres un certain nombre d'iteration, le sommet courant revient sur un sommet deja visite : on a alors un cycle.

Comme G est sans circuit, G possede donc une source s . Retirons s de G , on obtient alors le graphe $G - s$ qui est sans circuit (car c'est un sous graphe de G sans circuit) et possede k sommets. Par hypothese de recurrence, $G - s$ possede un ordre topologique σ . Ainsi, l'ordre $s \cdot \sigma$ est un ordre topologique de G . Cela acheve la recurrence et la preuve.

b. Effectuer un parcours en profondeur du graphe de la figure 1 depuis un sommet de votre choix et notez le temps t_{deb} de debut de traitement et le temps t_{fin} de fin de traitement pour chaque sommet.

Solution.

Revoyez le parcours en profondeur dans le livre Introduction a l'algorithmique de Cormen et al., si vous ne vous en souvenez plus. Pour rappel, le temps s'incremente a chaque traversee d'arete. Le temps de debut de traitement de u est celui auquel on decouvre le sommet u , le temps de fin de traitement est celui auquel on retourne a u apres avoir explore son dernier voisin.

Pour rappel, lorsqu'il n'y a plus de sommets a traiter dans la pile mais que tous les sommets n'ont pas encore ete visites, l'algorithme DFS choisi un nouveau sommet a explorer parmi ceux qui ne l'ont pas encore ete, jusqu'a ce que tous les sommets aient ete explores.

c. Classez les sommets par ordre de fin decroissant. Que remarquez vous ?

Solution. Cela donne un ordre topologique sur les sommets du graphe. Ce n'est pas particulier a cet exemple mais est toujours vrai, pour tout graphe sans circuit et tout sommet de depart du parcours.

d. Montrez que cela est une propriete generale : dans un graphe G qui satisfait les conditions de la question 1 de l'exo 2, quelque soit le sommet $u \in V$, l'ordre inverse de fin de traitement des sommets dans un parcours en profondeur de source u est un tri topologique.

N.B. : on utilise la version du parcours en profondeur qui ne s'arete que lorsque tout les sommets du graphe ont ete visites.

Solution. Dans le parcours DFS d'un graphe sans cycle, il n'y a pas d'arc retour : le sommet v a l'extremite d'un arc uv traverse par DFS est soit blanc, soit noir. S'il est blanc, son traitement doit finir avant de pouvoir finir celui de u qui l'a decouvert. On a donc $fin(u) > fin(v)$. Et si v est noir, alors son traitement est deja fini, alors que celui de u ne l'est pas, et on a encore $fin(u) > fin(v)$. Comme cela est vrai pour tous les arcs uv du graphe, l'ordre inverse de l'ordre de fin satisfait bien la propriete d'un tri topologique.

On va concevoir un deuxieme algorithme pour le meme probleme, plus evident mais plus delicat a implementer que DFS.

Definition [Source et puits]

Une **source** dans un graphe orienté est un sommet qui n'a aucun voisin entrant. Un sommet qui n'a aucun arc sortant est appelé un **puits**.

e. Montrez qu'un graphe orienté sans circuit (DAG) possède au moins une source.

Solution. Nous l'avons déjà montré à la question 1 de l'exercice 2.

f. Montrez que si u est une source d'un DAG G , alors il existe un ordre topologique de G qui commence par u .

Solution. Il suffit de prendre un tri topologique de $G - s$, qui est sans cycle, et de rajouter s au début. Ainsi, la propriété d'un tri topologique est satisfaite pour tous les arcs de $G - s$ et pour tous ceux entre s et les sommets de $V \setminus \{s\}$.

g. Déduisez de la question précédente un algorithme pour déterminer un tri topologique d'un DAG G .

Solution. On peut procéder itérativement comme suit en partant d'un tri topologique qui ne contient aucun sommet :

1. déterminer l'ensemble S des sources de G
2. les placer dans un ordre quelconque à la suite du tri topologique qu'on est en train de former
3. retirer les sommets de S de G
4. recommencer sur $G - S$, jusqu'à ce que le graphe ne contiennent plus de sommets

h. Comment déterminez les sources d'un DAG sur ses listes d'adjacences? Quel est l'étape la plus coûteuse en temps dans l'algorithme que vous avez proposé à la question précédente? Quel est la complexité totale de cet algorithme?

Solution. Pour déterminer les sources d'un DAG sur ses listes d'adjacences il suffit de parcourir tous les sommets du graphe et de vérifier pour chacun si la liste de ses voisins entrants est vide. Cela prend un temps $O(n)$.

L'étape la plus coûteuse est celle de retirer les sources S du graphe G . Elle peut se faire simplement en temps $O(m)$ en marquant toutes les sources au préalable et en parcourant les listes d'adjacences du graphe, tout en retirant les sommets marqués. Implémenté ainsi, la complexité de cet algorithme est $O(nm)$.

i. Comment améliorez l'implémentation de cet algorithme pour obtenir une complexité linéaire.

Solution. On peut faire deux choses pour obtenir une complexité linéaire.

D'abord, on peut maintenir le degré entrant des sommets lorsqu'on retire les sources de leur liste d'adjacence, et placer tous les sommets dont le degré entrant tombe à zéro dans une liste. Ainsi, il n'est plus nécessaire de parcourir tous les sommets pour chercher les sources du graphe à l'étape suivante.

Ensuite, on peut diminuer le temps pour former le graphe $G - S$ à $O(\sum_{s \in S} d_{courant}^{\circ}(s))$, où $d_{courant}^{\circ}(s)$ est le degré de s dans le graphe courant. Pour cela, il faut augmenter les listes d'adjacence en stockant pour chaque voisin sortant v de chaque sommet u , un

pointeur vers l'endroit où se trouve u dans les voisins entrants de v . Ainsi, on a plus besoin de parcourir les listes d'adjacences de G pour retirer les sources, pour chaque source s et chaque voisin sortant v de s on sait où trouver s , pour le retirer, dans les voisins entrants de v (il faut aussi que les listes d'adjacences soient doublement chaînées pour effectuer les retraits en temps constant).

La question de savoir comment procéder pour mettre les pointeurs dans les listes d'adjacence du graphe de départ est un exercice en soi. Une indication : il faut utiliser la même technique que pour trier les listes d'adjacences d'un graphe en temps $O(n+m)$ (oui, c'est possible de faire ça!).

Exercice 3.

A star is born.

L'algorithme A^* , dérivé dans l'algorithme 1, est un algorithme de plus court chemin dans un graphe (éventuellement orienté) pondéré positivement qui trouve un chemin de la source s à la destination t . C'est une variante de l'algorithme de Dijkstra dont le but n'est pas de déterminer un chemin de s vers toutes les destinations, mais seulement vers la destination t donnée. L'enjeu est de minimiser la partie du graphe qui est explorée par l'algorithme afin de découvrir un tel chemin. Pour cela, on utilise une fonction h qui est donnée avec le graphe et qui donne une estimation (a priori sans garantie) pour chaque nœud u de la distance de u à t .

Algorithme 1 : L'algorithme $A^*(G,s,t,h)$

```

1 Un tableau dist de taille  $n$ ; Un tableau coul de taille  $n$ ;
2 pour  $u$  de 0 à  $n - 1$  faire
3    $\lfloor$   $dist[u] \leftarrow +\infty$ ;  $f[u] \leftarrow +\infty$ ; coul[ $u$ ]  $\leftarrow$  blanc;
4  $dist[s] \leftarrow 0$ ;  $f[s] \leftarrow dist[s] + h[s]$ ; coul[ $s$ ] = gris;  $Q \leftarrow \{s\}$ ;
5 tant que  $Q \neq \emptyset$  et non(trouve) faire
6    $u \leftarrow \min_f(Q)$ ;  $Q \leftarrow Q \setminus \{u\}$ ; coul[ $u$ ]  $\leftarrow$  noir;
7   si  $u = t$  alors
8      $\lfloor$  trouve  $\leftarrow$  vrai
9   sinon
10    pour  $v \in N(u)$  faire
11      si  $dist[u] + w(u, v) < dist[v]$  alors
12         $dist[v] \leftarrow dist[u] + w(u, v)$ ;
13        si coul[ $v$ ] = blanc alors
14           $Q \leftarrow Q \cup \{v\}$ ; coul[ $v$ ]  $\leftarrow$  gris;

```

a. Exécutez l'algorithme $A^*(G, s, t, h)$ sur le graphe de la figure 2 en prenant pour h la distance euclidienne sur le dessin fourni, le nœud bleu pour s et le nœud rouge pour t . Les arêtes de G ont toutes poids 1.

Solution. Prenez votre temps...

b. Analysez le pseudo-code de l'algorithme 1. Quelle est la différence entre A^* et l'algorithme de Dijkstra ?

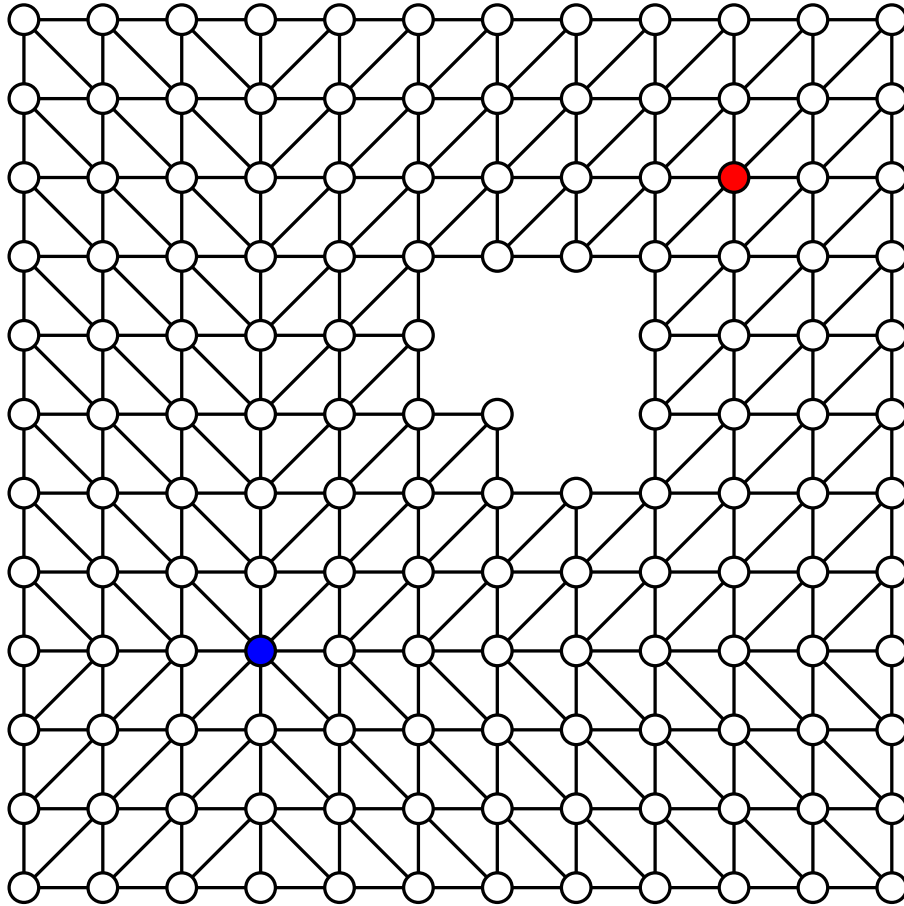


FIGURE 2 – Un graphe exemple pour l'exécution de $A^*(G, s, t, h)$ avec un poids unitaire sur toutes les arêtes. Prendre pour h la distance euclidienne sur le dessin fourni, le noeud bleu pour s et le noeud rouge pour t .

Solution. La différence est que dans A^* , le prochain noeud explore u n'est pas celui qui minimise la distance à la source $dist[u]$, mais qui minimise $f(u) = dist[u] + h[u]$, c'est à dire la longueur estimée de tout le chemin de s à t passant par u . C'est ce qui favorise les plus courts chemins vers t , alors que Dijkstra explore le graphe dans toutes les directions.

On considère maintenant des fonctions d'estimation h ayant la propriété de ne jamais surestimer la distance de u à t , c'est à dire telles que $\forall u \in V, h(u) \leq dist(u, t)$.

c. Montrez que pour de telles h , le chemin de s à t qui est découvert par A^* est bien un plus court chemin de s à t .

Indication. Considérez le moment où t est choisi comme le noeud u à la ligne 6.

Solution. Par l'absurde, supposons que lorsque t est choisi comme u par l'algorithme à la ligne 6, $dist[t] \neq dist(s, t)$. Nécessairement, on a alors $dist[t] > dist(s, t)$, car, comme dans BFS , il existe un chemin de longueur $dist[t]$ de s à t . Considérons un plus court chemin P de s à t , de longueur $dist(s, t) < dist[t]$ et considérons le premier sommet v non noir sur ce chemin (s est noir). v est gris car il a un voisin noir.

Montrons, par l'absurde encore, que sur le sous-chemin P_{sv} de P de s à v , tous les noeuds w satisfont $dist[w] = dist(s, w)$. Supposons que ce ne soit pas le cas et considérons le noeud v' le plus proche de s sur P_{sv} tel que $dist[w] > dist(s, w)$. $v' \neq s$ car $dist[s] = 0 = dist(s, s)$. Ainsi, v' a un predecesseur v'' sur P_{sv} , qui est noir et tel que $dist[v''] = dist(s, v'')$. Donc, lorsque l'arc $v''v'$ a été relâché depuis v'' , v' a reçu sa distance correcte depuis s . En conclusion, tous les noeuds de P , y compris v , satisfont ont reçu leur distance correcte depuis s .

Ainsi, lorsque t est choisi comme u à la ligne 6, on a $f(v) = dist[v] + h[v] = dist(s, t) < dist[t] = f(t)$ et l'algorithme aurait dû choisir v au lieu de t . C'est donc que contrairement à ce qu'on a supposé, lorsque t est choisi comme u à la ligne 6, on a bien $dist[t] = dist(s, t)$.

d. A-t-on $dist[u] = dist(s, u)$ pour tous les noeuds noirs u ?

Solution. Non, un noeud u peut être exploré car sa distance à t est trop largement surestimée, sans que ses voisins soient nécessairement explorés. Ainsi, u peut très bien ne pas recevoir sa vraie distance depuis s , car il aurait fallu explorer ses voisins pour la découvrir.

e. Dans le cas (très) particulier où $h(u) = dist(u, t)$, quels sont les noeuds colorés en noirs à la fin de l'algorithme ?

Solution. Dans ce cas là, seuls les noeuds sur un plus court chemin entre s et t ont une chance d'être explorés, car les autres ont une plus grande valeur de f . Lesquels vont être effectivement explorés dépend de comment les égalités vont être arbitrés parmi les sommets u qui réalisent le minimum de la fonction f à la ligne 6. En résumé la réponse est un sous-ensemble A des noeuds qui se trouvent sur un plus court chemin de s à t et tel que A contient au moins un plus court chemin de s à t .

Exercice 4.

Pirate de grand chemin

La guérilla fait rage dans la région de Paradisio. L'armée rebelle prévoit de s'emparer de la ville de DolceVita. Pour maximiser ses chances de réussite, elle veut éviter d'une réaction rapide du pouvoir en place, qui concentre ses forces à Paradisio. La carte de la région, avec les différents temps de trajet, est donnée sur la figure 3. Les rebelles possèdent l'arsenal nécessaire pour couper une voie de communication, mais pas plus, et ils souhaitent rallonger au maximum le temps de trajet de Paradisio à DolceVita.

a. Quelle est la voie que l'armée rebelle doit couper pour maximiser le temps de trajet de Paradisio à DolceVita ?

Solution.

b. Dans le cas général, étant donné un graphe G pondéré positivement et deux sommets A et B , proposer un algorithme simple pour trouver l'arête à retirer de G afin d'augmenter au maximum la longueur du plus court chemin entre les sommets A et B .

Solution.

c. Quelle est la complexité de votre algorithme ?

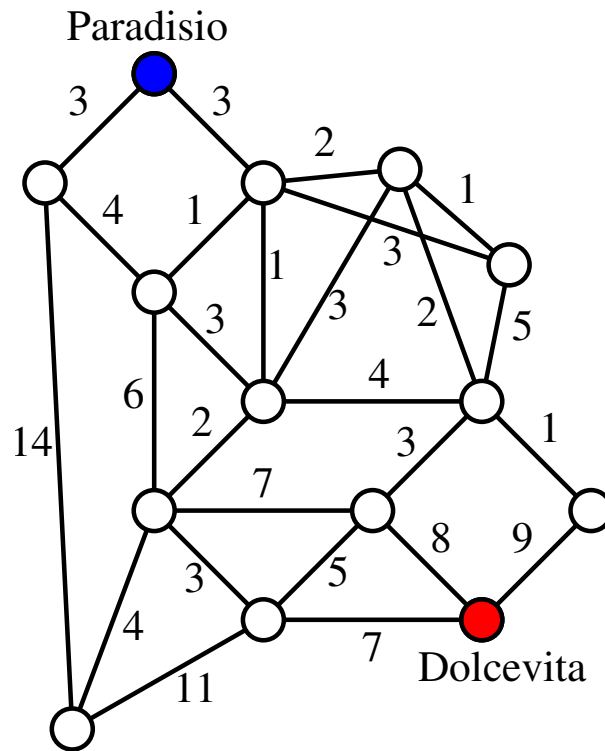


FIGURE 3 – La carte de la region de Paradisio avec les temps de trajets entre villes.

Solution.

d. Proposez un algorithme lineaire qui decide s'il est possible d'augmenter la longueur du plus court chemin entre A et B en retirant une seule arete de G .

Indication. Adapter un peu l'algorithme de Dijkstra pour qu'il garde trace de tous les plus courts chemin entre la source et les autres sommets.

Solution.