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 traite sur une seance d'1h30. Il est recommande que vous utilisiez les exercices que vous n'avez pas eu le temps de faire en seance comme entrainement en vue de l'examen final. Les exercices les plus importants sont les 1, 2 et 3 : commencez par ceux la.

Exercice 1. *Differentes facons 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 a v contient un chemin Q de a a b qui n'est pas un plus court chemin de a a b. Alors en remplacant Q par un plus court chemin Q' de a a b dans P, on obtient un chemin $P' = P_1 \cdot Q' \cdot P_2$ de u a v qui est plus court que le chemin P.

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

Solution. Si wv est la derniere arete d'un plus court chemin de u a v alors la premiere partie P de ce chemin est un plus court chemin de u a w, d'apres la question precedante. Reciproquement, si P' est un plus court chemin de u a w, alors $P' \cdot wv$ est un plus court chemin de u a v, car sa longueur est bien la distance de u a v. Ainsi, les plus courts chemins de u a v terminant par wv sont en meme nombre que les plus courts chemins de u a 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 derniere arete d'un plus court chemin de u a v. Exprimez $Nbc_u(v)$ en fonctions des $Nbc_u(w)$ pour $w \in Prec_u(v)$.

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'apres la question precedante.

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 precedante, 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 $p_1(v), p_2(v), \ldots$ 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 $[\![1, Nbc_u(v)]\!]$. 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 (oriente) 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.

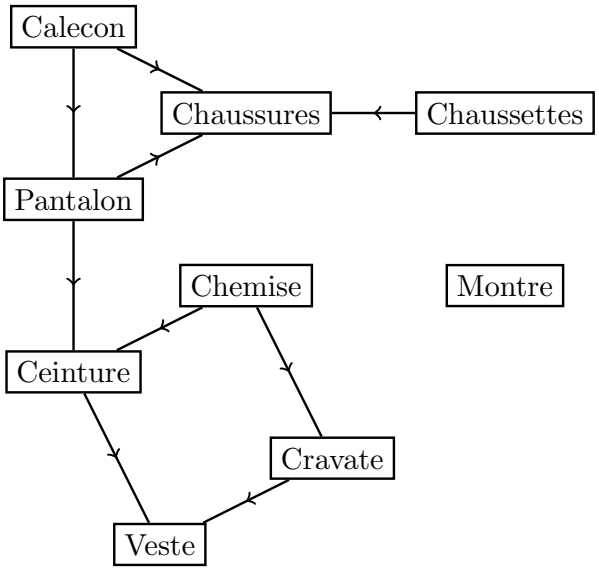

FIGURE 1 – Le graphe des contraintes sur l'ordre d'habillage. Un arc du vetement A vers le vetement B signifie que le vetement A doit obligatoirement etre mis avant de mettre le vetement B.

On se propose de trouver une méthode systematique pour habiller tout le monde, quels que soient ses vetements. C'est a dire, qu'etant 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 circut, 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 u 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 tritement 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'odre inverse de l'ordre de fin satisafit 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 oriente est un sommet qui n'a aucun voisin entrant. Un sommet qui n'a aucun arc sortant est appele un **puits**.

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

Solution. Nous l'avons deja montre a 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 $G - s$, quie st sans cycle, et de rajouter s au debut. Aisni, la propriete 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. Deduisez de la question precedante un algorithme pour determiner un tri topologique d'un DAG G.

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

1. determiner l'ensemble S des sources de G
2. les placer dans un ordre quelconque a 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'a ce que le graphe ne contiennent plus de sommets

h. Comment determinez les sources d'un DAG sur ses listes d'adjacences ? Quel est l'etape la plus couteuse en temps dans l'algorithme que vous avez propose a la question precedante ? Quel est la complexite totale de cet algorithme ?

Solution. Pour determiner les sources d'un DAG sur ses listes d'adjacences il suffit de parcourir tous les sommets du graphe et de verifier pour chacun si la liste de ses voisins entrants est vide. Cela prend un temps $O(n)$.
L'etape la plus couteuse 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 prealable et en parcourant les listes d'adjacences du graphe, tout en en retirant les sommets marques.
Implemente ainsi, la complexite de cet algorithme est $O(nm)$.

i. Comment ameliorez l'implementation de cet algorithme pour obtenir une complexite lineaire.

Solution. On peut faire deux choses pour obtenir une complexite lineaire.
D'abord, on peut maintenir le degre entrant des sommets lorsqu'on retire les sources de leur liste d'adjacence, et placer tous les sommets dont le degre entrant tombe a zero dans une liste. Ainsi, il n'est plus necessaire de parcourir tous les sommets pour chercher les sources du graphe a l'etape suivante.
Ensuite, on peut diminuer le temps pour former le graphe $G-S$ a $O(\sum_{s\in S} d^{\circ}_{courant}(s))$, ou $d^{\circ}_{courant}(s)$ est le degre 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 ou 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 ou trouver s, pour le retirer, dans les voisins entrants de v (il faut aussi que les listes d'adjacences soient doublement chainees pour effectuer les retraits en temps constant).

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

Exercice 3. *A star is born.*

L'algorithme A*, derit dans l'algorithme 1, est un algorithme de plus court chemin dans un graphe (eventuellement oriente) pondere positivement qui trouve un chemin de la source s a la destination t. C'est une variante de l'algorithme de Dijkstra dont le but n'est pas de determiner un chemin de s vers toutes les destinations, mais seulement vers la destination t donnee. L'enjeu est de minimiser la partie du graphe qui est explore par l'algorithme afin de decouvrir un tel chemin. Pour cela, on utilise une fonction h qui est donnee avec le graphe et qui donne une estimation (a priori sans garantie) pour chaque noeud u de la distance de u a 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 a n − 1 faire
3      dist[u] ← +∞ ; f[u] ← +∞ ; coul[u] ← blanc;
4  dist[s] ← 0 ; f[s] ← dist[s] + h[s] ; coul[s] = gris ; Q ← {s};
5  tant que Q ≠ ∅ et non(trouve) faire
6      u ← min_f(Q) ; Q ← Q \ {u} ; coul[u] ← noir;
7      si u = t alors
8          trouve ← vrai
9      sinon
10         pour v ∈ N(u) faire
11             si dist[u] + w(u, v) < dist[v] alors
12                 dist[v] ← dist[u] + w(u, v) ; f[v] ← dist[v] + h(v);
13                 si coul[v]=blanc alors
14                     Q ← Q ∪ {v} ; coul[v] ← gris;
```

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

Solution. Prenez votre temps...

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

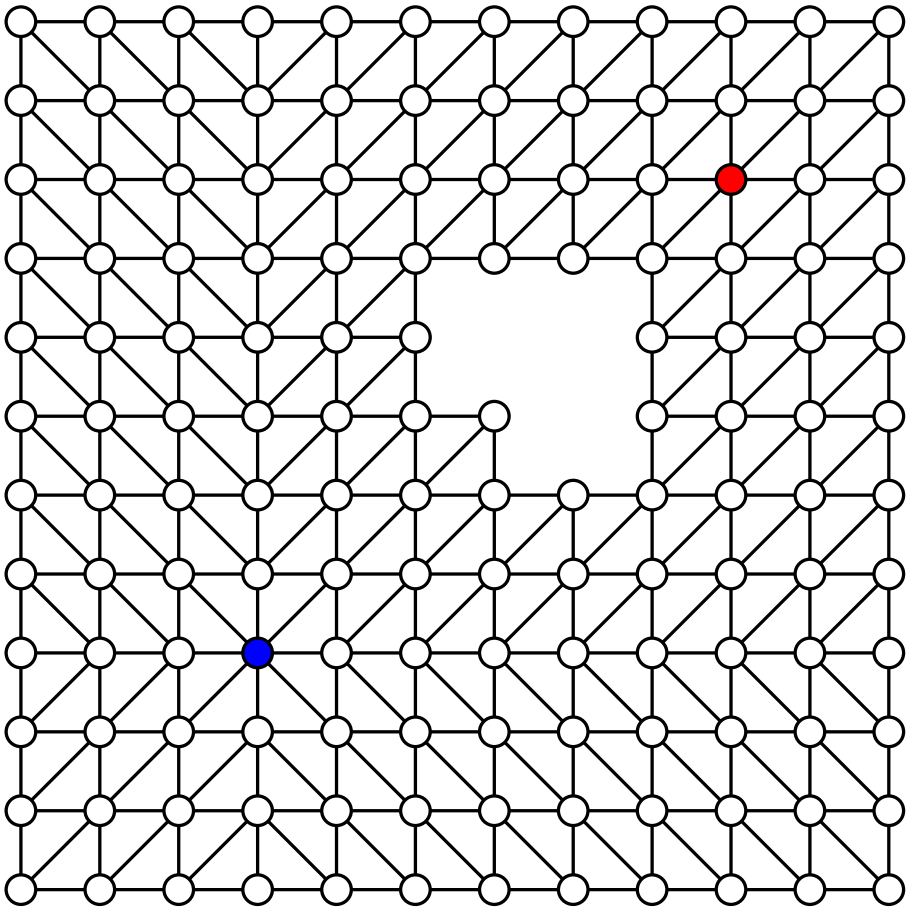

FIGURE 2 – Un graphe exemple pour l'execution de $A*(G, s, t, h)$ avec un poids unitaire sur toutes les aretes. Prendre pour h la distance euclidienne sur le dessin fourni, le noeud bleu pour s et le noeud rouge pour t.

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

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

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

Indication. Considerez le moment ou t est choisi comme le noeud u a la ligne 6.

Solution. Par l'absurde, supposons que lorsque t est choisi comme u par l'algorithme a la ligne 6, $dist[t] \neq dist(s, t)$. Necessairement, on a alors $dist[t] > dist(s, t)$, car, comme dans BFS, il existe un chemin de longueur $dist[t]$ de s a t. Considerons un plus court chemin P de s a t, de longueur $dist(s, t) < dist[t]$ et considerons 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 a v, tous les noeuds w satisfont $dist[w] = dist(s, w)$. Supposons que ce ne soit pas le cas et considerons le neoud 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 ete relache depuis v'', v' a recu sa distance correcte depuis s. En conclusion, tous les noeuds de P, y compris v, satisfont ont recu leur distance correcte depuis s.

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

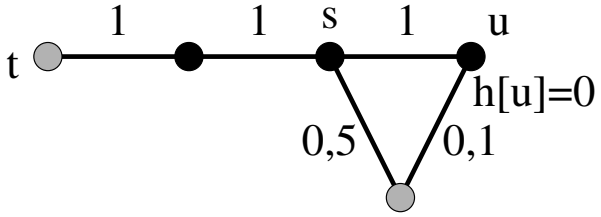

FIGURE 3 – Un graphe G, une source s et une destination t, pour lesquels A* colore le sommet u en noir sans lui affecter sa distance correcte. Pour tous les sommets v differents de u on a $h(v) = dist(v, t)$, et pour u on a $h(u) = 0$. Le sommet u devient noir au premier tour de l'algo car il minimise $f(u) = dist[u] + h(u)$. La distance de s a u est $0,6$ mais l'algo lui affecte $dist[u] = 1$ car le voisin commun a u et s n'est jamais explore (sa valeur de f est 3, la plus grande parmi les sommets de G).

Solution. Non, un noeud u peut etre explore car sa distance a t est trop largement sous-estimee, sans que ses voisins soient necessairement explores. Ainsi, u peut tres bien ne pas recevoir sa vraie distance depuis s, car il aurait fallu explorer ses voisins pour la decouvrir. Voir l'exemple de la figure 3.

e. Dans le cas (tres) particulier ou $h(u) = dist(u, t)$, quels sont les noeuds colores en noirs a la fin de l'algorithme ?

Solution. Dans ce cas la, seuls les noeuds sur un plus court chemin entre s et t ont une chance d'etre explores, car les autres ont une plus grande valeur de f. Lesquels vont etre effectivement explores depend de comment les egalites vont etre arbitres parmi les sommets u qui realisent le minimum de la fonction f a la ligne 6. En conclusion, la reponse est un sous ensemble A des neouds qui se trouvent sur un plus court chemin de s a t et tel que A contient au moins un plus court chemin de s a t.

Exercice 4. *Bandit de grand chemin*

La guerilla fait rage dans la region de Paradisio. L'armee rebelle prevoit de s'emparer de la ville de Dolcevita. Pour maximiser ses chances de reussite, elle veut eviter d'une reaction rapide du pouvoir en place, qui concentre ses forces a Paradisio. La carte de la region, avec les differents temps de trajet, est donnee sur la figure 4. Les rebelles possede l'arsenal necessaire pour couper une voie de communication, mais pas plus, et ils souhaitent rallonger au maximum le temps de trajet de Paradisio a Dolcevita.

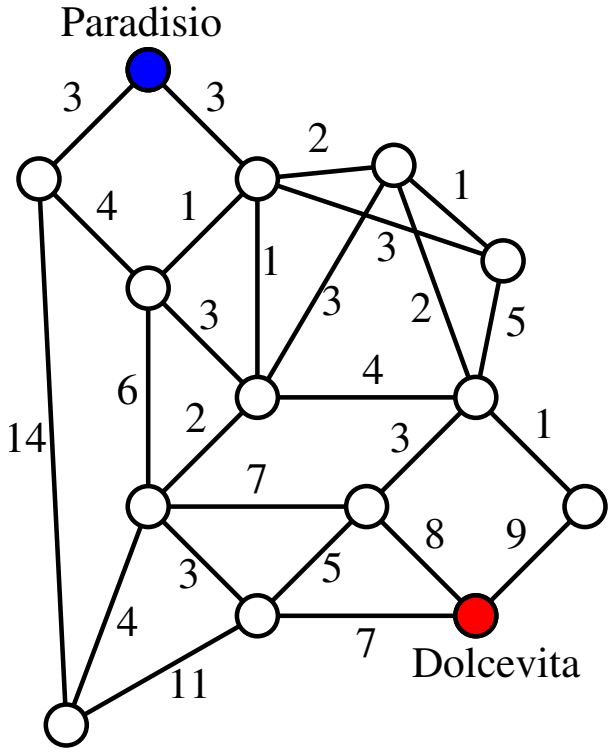

FIGURE 4 – La carte de la region de Paradisio avec les temps de trajets entre villes.

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

Solution. Il faut couper la voie de longueur 3 au sud est de Paradisio. On note e cette arete. En effet, la distance entre Paradisio et Dolcevita est 16. Il existe un unique chemin P de longueur 16 entre Paradisio et Dolcevita. Si on veut augmenter la distance, il faut et il suffit de retirer une arete sur ce chemin P. Or il se trouve qu'apres P, le deuxieme chemin P' le plus court entre entre Paradisio et Dolcevita est aussi unique et a pour longueur 17. De plus, la seule arete en commun entre P et P' est l'arete e. Ainsi, si on decide de retirer une arete de P autre que e, la distance entre les deux villes augmente exactement de 1. Alors que si on decide de retirer e la distance augmente d'au moins 2 (il faudrait calculer la longueur du 3eme chemin le plus court pour savoir de combien elle augmente exactement). Il faut donc choisir de couper la voie e de longueur 3 au sud est de Paradisio.

b. Dans le cas general, etant donne un graphe G pondere positivement et deux sommets A et B, proposer un algorithme simple pour trouver l'arete a retirer de G afin

d'augmenter au maximum la longueur du plus court chemin entre les sommets A et B.

Solution. On peut executer $m+1$ fois l'algorithme de Dijkstra avec source A : une fois pour chaque arete e, en retirant uniquement l'arete e, puis une fois sans retirer aucune arete. Ainsi, on peut determiner quel retrait d'arete augmente le plus la distance et de combien il l'augmente (s'il l'augmente strictement).

c. Quelle est la complexite de votre algorithme ?

Solution. La complexite de l'algorithme propose a la question precedente est m fois celle de Dijkstra, c'est a dire $O(m^2 + mn \log n)$ en implementant la file de priorite de l'algorithme de Dijkstra avec un tas de Fibonacci.

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. On peut commencer par executer Dijkstra depuis A en l'augmentant comme vu pour le BFS a l'exo 1, de sorte a ce que chaque sommet v garde la liste des sommets u pour lesquels la relaxation de l'arete uv produit la distance minimum de A a v. Ainsi, en remontant a partir de B jusqu'a A grace a ces listes, on obtient le graphe G_{AB} induit par tous les plus courts chemins de A a B.

Il faut ensuite remarquer qu'il est possible d'augmenter la longueur du plus court chemin de A a B en ne retirant qu'une arete ssi tous les plus courts chemins de A a B partagent une arete e. C'est a dire que e est un **isthme** du graphe G_{AB} [1].

Ainsi, il reste a faire un algorithme pour lister les isthmes d'un graphe, ce qui est connu pour etre faisable en temps lineaire pour un graphe quelconque G'. A cette fin, on commence par faire un parcours en profondeur de G' a partir d'une source s quelconque. Tout isthme de G' est necessairement une arete de l'arbre DFS note T. De plus une arete $x\,pere(x)$ de T est un isthme de G ssi aucun des sommets descendant de x dans T n'a un arc retour vers un sommet hors du sous arbre T_x enracine en x.

Cela se verifie en faisant un parcours de T depuis ses feuilles jusqu'a la racine et en determinant pour chaque sommet $u \in T$ quel est le minimum $deb_{min}(u)$ de la date de debut de traitement dans le DFS des sommets z tels que $\exists v \in T_u$ tel que vz est un arc retour du DFS. On a alors, $x\,pere(x)$ est un isthme ssi $deb_{min}(x) < deb(x)$. Cela prend un temps $O(n + m)$.

Comme chaque etape de l'algo decrit ci-dessus prend un temps $O(n + m)$, le temps total pour determiner s'il existe une arete dont le retrait augment la distance de A a B est $O(n + m)$.

1. Un isthme dans un graphe est une arete dont le retrait deconnecte le graphe