ÉCOLE NORMALE SUPÉRIEURE DE LYON

Concours d'admission session 2024

Filière universitaire : Second concours

COMPOSITION D'INFORMATIQUE

Durée : 3 heures

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve.

⋆ ⋆ ⋆

Déplacements dans des graphes

Ce sujet est découpé en 3 parties portant toutes sur les graphes : les deux premières portent sur le problème du plus court chemin dans un graphe pondéré, la troisième étudie des valuations sur des graphes non pondérés. Toutes les définitions sont rappelées en préambule.

Les deux premières parties se suivent, la troisième est indépendante.

Définitions et notations

Notations. Pour un ensemble fini X, on désignera par $|X|$ le cardinal de X.

Un **graphe orienté** est un couple $G = (S, A)$ où
- S est un ensemble fini dont les éléments sont appelés les **sommets** de G,
- A est une partie de $S \times S$ dont les éléments sont appelés les **arcs** de G.

On pose le vocabulaire suivant :
- Pour un arc $(x, y) \in A$, x est appelé **origine** et y est appelé **destination**.
- Un **voisin** d'un sommet x est un sommet y tel qu'on ait $(x, y) \in A$.
- Le **degré sortant** d'un sommet $x \in S$, noté $d^+(x)$, est le nombre de voisins de x : $d^+(x) = |\{y; (x, y) \in A\}|$.
- Un **chemin** c dans G est une suite finie non vide $(x_0, x_1, \ldots, x_n)$ de sommets de G telle que pour tout $i \in \{1, \ldots, n\}$ on a $(x_{i-1}, x_i) \in A$. L'entier n est la **longueur** du chemin c, on le note aussi $|c|$.
- Pour un chemin $c = (x_0, x_1, \ldots, x_n)$, x_0 est appelé **origine** et x_n est appelé **destination** (comme pour les arcs).
- Pour deux chemins $c = (x_0, \ldots, x_n)$ et $d = (y_0, \ldots, y_m)$ tels que la destination de c est l'origine de d (donc $x_n = y_0$), on note $c.d$ la concaténation de c et d, définie comme $c.d = (x_0, \ldots, x_n, y_1, \ldots, y_m)$. C'est un chemin de longueur $|c| + |d|$.
- Un **cycle** est un chemin dont l'origine et la destination sont le même sommet.

On ne considèrera ici que des graphes **sans boucles**, c'est-à-dire que pour chaque arc, l'origine et la destination sont des sommets distincts.

Un **graphe orienté à arcs pondérés** (dans la suite on parlera simplement de **graphe pondéré**) est un triplet $G = (S, A, p)$ où
- (S, A) est un graphe orienté,
- p est une fonction appelée **pondération** qui associe à chaque arc $a \in A$ un nombre $p(a)$ appelé **poids** de a.
- Le **poids** d'un chemin $c = (x_0, x_1, \ldots, x_n)$ est la somme des poids des arcs qui le constituent : $p(c) = \sum_{i=1}^{n} p(x_{i-1}, x_i)$.

L'ensemble de nombres dans lequel p prend ses valeurs (réelles, entières, ou autres) sera précisé lorsque ce sera utile.

Complexité. Le **temps d'exécution** d'un algorithme pour une entrée donnée est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, ou autres en fonction du contexte) nécessaires à l'exécution de l'algorithme. La **complexité** d'un algorithme exprime, en

fonction de certains paramètres pertinents (taille de la donnée d'entrée, nombre de sommets ou d'arcs pour un graphe, etc) le temps d'exécution de l'algorithme dans le pire des cas, à valeur fixée des paramètres. Pour une complexité qui dépend de paramètres $n_1, \ldots, n_k$, on dira qu'un algorithme a une complexité en $O(f(n_1, \ldots, n_k))$, lorsqu'il existe des constantes K et $N_1, \ldots, N_k$ telles que la complexité soit au plus $K \times f(n_1, \ldots, n_k)$ pour tous $n_1 \geq N_1, \ldots, n_k \geq N_k$.

Lorsqu'il est demandé de d'évaluer une certaine complexité, la réponse est attendue en O et le candidat devra justifier soigneusement la réponse proposée.

Écriture d'algorithmes. Dans ce sujet, les algorithmes sont écrits en français et l'indentation est utilisée pour mettre en évidence les parties des algorithmes concernées par les différentes structures de contrôle (comme dans le langage Python). La notation $x \leftarrow e$ représente l'affectation à la variable x du résultat de l'évaluation de e.

Quand un algorithme ou morceau d'algorithme est demandé, celui-ci pourra être donné soit en français avec des conventions similaires, soit dans un langage de programmation au choix du candidat.

Partie I. Étude de l'algorithme de Dijkstra

L'algorithme de Dijkstra détermine le poids minimal des chemins d'un sommet s aux autres sommets d'un graphe pondéré par une fonction **positive**. Il utilise deux sous-ensembles E et F des sommets du graphe. Une estimation du poids minimal d'un chemin entre s et chaque sommet x est maintenue dans l'élément $D[x]$ d'un tableau D. À chaque itération, un des sommets $u \in F$ minimisant D et qui n'est pas dans E est sélectionné, il est retiré de F, ajouté à E, et les estimations des voisins sont mises à jour par une procédure appelée **relâchement**. L'algorithme est présenté en figure 1. Deux opérations auxiliaires y sont utilisées :

extraire renvoie un sommet u élément de $F \setminus E$ tel que $D[u]$ soit minimal,

relâcher (u, v) remplace $D[v]$ par $D[u] + p(u, v)$ si cette valeur est inférieure.

On laisse l'implémentation de ces opérations non spécifiée pour le moment.

Question 1. Expliquer pourquoi l'algorithme de Dijkstra se termine toujours.

Question 2. Expliquer pourquoi on peut représenter les deux ensembles E et F au moyen d'un unique tableau de booléens. Justifier qu'avec une telle représentation, la complexité totale des appels à « extraire » est au plus quadratique en le nombre de sommets de G.

Question 3. Estimer le nombre d'opérations nécessaire à l'exécution de l'algorithme si le graphe est représenté par une matrice d'adjacence, en précisant quelles sont les opérations élémentaires choisies. On considèrera qu'obtenir le poids d'un arc connaissant ses extrémités se fait en temps constant et que l'accès à un élément du tableau D se fait aussi en temps constant.

Question 4. Proposer des structures de données adaptées à la représentation du graphe et de l'ensemble F pour obtenir la meilleure complexité possible. Préciser la complexité obtenue,

Algorithme Dijkstra (G, s).

Entrées : un graphe pondéré $G = (S, A, p)$,
un sommet $s \in S$
Résultat : le tableau des poids minimaux depuis s

$E \leftarrow \emptyset$
$F \leftarrow S$
Pour chaque $x \in S$, faire
    $D[x] \leftarrow +\infty$
$D[s] \leftarrow 0$
Tant que F n'est pas vide, faire
    $u \leftarrow$ extraire,
    retirer u de F,
    ajouter u à E,
    pour chaque v voisin de u,
        relâcher (u, v).
Renvoyer D.

FIGURE 1 – Trame de l'algorithme de Dijkstra

en fonction du nombre de sommets et d'arcs du graphe. On fera les mêmes hypothèses qu'en question 3 sur l'accès aux poids et au tableau D.

Partie II. Un autre algorithme

L'algorithme de Dijkstra permet de calculer le poids minimal des chemins allant d'un sommet s à tous les autres sommets accessibles à partir de s. Une des hypothèses pour que l'algorithme de Dijkstra soit correct est que les poids soient tous positifs. Lorsque les pondérations peuvent être négatives, d'une part il n'existe pas forcément de plus court chemin, d'autre part l'algorithme de Dijkstra peut ne pas en trouver même s'il en existe.

Question 5. Donner un exemple de graphe pondéré (avec certaines pondérations négatives) et d'exécution de l'algorithme de Dijkstra (tel que présenté en figure 1) qui illustre la non correction de l'algorithme, dans un cas où les plus courts chemins existent.

Question 6. Donner un exemple de graphe pondéré pour lequel il n'existe pas de chemin de poids minimal entre s et un autre sommet accessible à partir de s.

Jusqu'à la fin de cette partie, on suppose qu'il existe un chemin de poids minimal de s à chacun des autres sommets du graphe (il n'y a pas de cycle de poids négatif, un tel cycle serait dit **absorbant**). On note $D_{\min}(x)$ le poids minimal d'un chemin de s à x.

Question 7. Démontrer que $D_{\min}$ vérifie l'équation suivante :

$$\begin{cases} D_{\min}(v) = \min_{(u,v)\in A}\Big(D_{\min}(u) + p(u,v)\Big) & \text{pour tout } v \in S \setminus \{s\} \\ D_{\min}(s) = 0 \end{cases}$$

On considère maintenant l'algorithme de la figure 2.

Algorithme Poids minimal (G, s).

Entrées : un graphe pondéré $G = (S, A, p)$,
un sommet $s \in S$
Résultat : le tableau des poids minimaux depuis s

Pour chaque $x \in S \setminus \{s\}$, faire
    $D[x] \leftarrow +\infty$
$D[s] \leftarrow 0$
Tant qu'il existe un arc (u, v) tel que $D[v] > D[u] + p(u, v)$, faire
    $D[v] \leftarrow \min\big(D[v], D[u] + p(u, v)\big)$.
Renvoyer D.

FIGURE 2 – Algorithme alternatif de calcul des chemins de poids minimal

---

Question 8. Exécuter l'algorithme sur le graphe de la figure 3. On notera la séquence des arcs choisis et la suite des vecteurs D obtenus à la fin de chaque itération de la boucle **tant que**.

Question 9. Démontrer que si l'algorithme se termine, alors le vecteur D renvoyé est solution de l'équation de la question 7.

Question 10. Démontrer que l'algorithme se termine, toujours sous l'hypothèse qu'il n'y a pas de cycle absorbant. On pourra chercher un variant de l'itération en utilisant le fait que l'on peut se limiter à un ensemble fini de chemins pour trouver ceux de poids minimal.

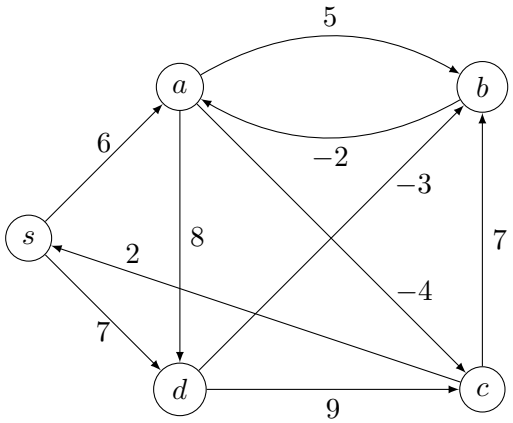

FIGURE 3 – Graphe pondéré à traiter en question 8

La correction de l'algorithme de la figure 2 découle des questions 9 et 10 : l'exécution se termine toujours et le tableau renvoyé donne les poids minimaux des chemins d'origine s.

Question 11. Modifier l'algorithme afin d'avoir le routage, c'est à dire, pour chaque sommet x, un chemin de poids minimal de s à x.

On remplace la boucle **tant que** de l'algorithme par les instructions suivantes :

Répéter $|S| - 1$ fois
    pour chaque $(u, v) \in A$, faire
        $D[v] \leftarrow \min(D[v], D[u] + p(u, v))$

Question 12. En supposant toujours qu'il n'y a pas de cycle absorbant, montrer que l'algorithme ainsi modifié est correct. En déduire que la complexité de l'algorithme de la figure 2 peut être réduite à $O(|S|.|A|)$.

Pour finir, on ne suppose plus que le graphe considéré n'a pas de cycle absorbant et on s'intéresse au comportement de l'algorithme dans ce cas plus général.

Question 13. Que se passe-t-il si le graphe possède un cycle absorbant ? Modifier l'algorithme pour qu'il détecte cette situation : en prenant en entrée un graphe pondéré (S, A, p) et un sommet $s \in S$, l'algorithme modifié devra soit répondre qu'il y a un cycle absorbant si c'est le cas (on ne demande pas de donner un tel cycle, seulement de détecter qu'il en existe), soit renvoyer le tableau des poids minimaux depuis s.

Partie III. Tas de sable sur un graphe

Dans cette partie, on étudie le modèle dit du **tas de sable**, qui décrit un processus d'éboulement sur un graphe. Chaque sommet du graphe est garni d'un certain nombre de grains de sable. À chaque étape du processus, on choisit un sommet qui contient au moins autant de grains qu'il a de voisins et ce sommet passe un grain de sable à chacun de ses voisins. La figure 4 montre un exemple de trois éboulements successifs sur un graphe donné, le sommet choisi pour la prochaine étape est indiqué en gras.

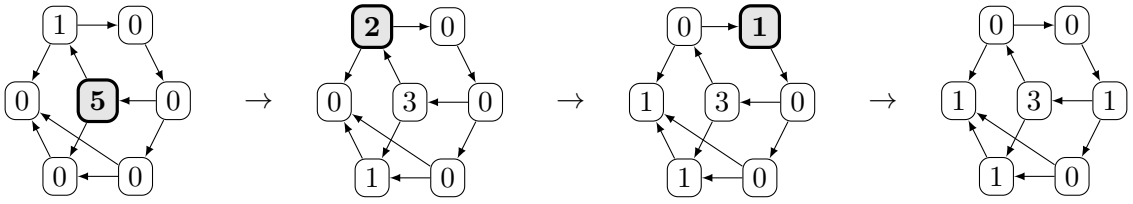

FIGURE 4 – Exemple de suite d'éboulements

Formellement, on considère un graphe orienté $G = (S, A)$ sans boucle. On appelle **configuration** une fonction $k : S \to \mathbb{N}$. On appelle **effet** une fonction $f : S \to \mathbb{Z}$. Pour un sommet x,

on définit l'effet δ_x de la façon suivante :

$$\delta_x(y) = \begin{cases} -d^+(x) & \text{si } y = x \\ +1 & \text{si } (x,y) \in A \\ 0 & \text{si } y \neq x \text{ et } (x,y) \notin A \end{cases}$$

La fonction δ_x représente l'effet de l'éboulement du sommet x tel que décrit plus haut : si k est une configuration avec $k(x) \geq d^+(x)$, alors $k + \delta_x$ est la configuration obtenue en enlevant $d^+(x)$ grains à x et en les ajoutant à ses voisins.

On note $k \to k'$ pour signifier qu'il existe une transition de la configuration k à la configuration k', c'est-à-dire qu'il existe sommet x avec $d^+(x) \neq 0$ tel que $k(x) \geq d^+(x)$ et $k' = k + \delta_x$. On note $\to^*$ la clôture réflexive et transitive de $\to$, c'est-à-dire que $k \to^* k'$ lorsqu'il existe une suite de transitions $k = k_0 \to k_1 \to \cdots \to k_n = k'$ avec $n \geq 0$.

Question 14. Montrer que si, dans une configuration k donnée, il est possible de faire des éboulements en deux sommets distincts x et y, alors il est possible de faire les deux successivement et que la configuration obtenue ne dépend pas de l'ordre dans lequel on les fait.

Question 15. Montrer que si on a deux suites de transitions $k \to^* a$ et $k \to^* b$ partant de la même configuration k, il existe une configuration k' telle que $a \to^* k'$ et $b \to^* k'$.

Une configuration k est dite **stable** si aucun éboulement ne peut s'y produire, c'est-à-dire s'il n'y a aucune transition $k \to k'$ partant de k, autrement si pour tout sommet x tel que $d^+(x) \neq 0$ on a $k(x) < d^+(x)$.

Question 16. Donner un exemple de graphe et de configuration sur ce graphe à partir de laquelle il est possible de faire une suite infinie de transitions.

Question 17. Montrer que si le graphe G est sans cycle, alors il n'existe pas de suite infinie de transitions.

On appelle **puits** un sommet dont le degré sortant est nul. Dans notre situation, les puits sont les sommets d'où il n'y a jamais d'éboulement.

Question 18. Montrer que si pour tout sommet x il existe au moins un puits accessible depuis x, alors il n'existe pas de suite infinie de transitions.

Question 19. En déduire que dans ce cas, pour toute configuration k, il y a exactement une configuration stable $\hat{k}$ telle que $k \to^* \hat{k}$.

* *
*