Corrigé

Examen de Théorie des Graphes

EPITA ING1, S6 2016; A. DURET-LUTZ

Durée : 1 heure 30

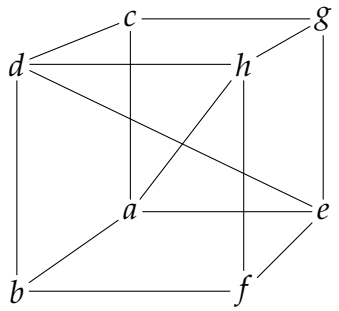
mars 2016

Consignes

— Cet examen se déroule **sans document** et **sans calculatrice**. Vous pouvez toutefois vous servir d'un dénoyauteur de câpres à condition de ne pas déranger vos voisins.
— Il y a 4 pages d'énoncé. Répondez sur le sujet dans les cadres prévus à cet effet, sans donner plus de détails que ceux nécessaires à vous justifier.

1 Ionis Portal (8 points)

1. **(2 pt)** Le Général Jack O'Neill peut se déplacer aux 8 coins de l'univers (qui est cubique) grâce à un système de portails permettant de voyager rapidement entre deux points. La carte des connexions entre ces portails est donnée ainsi :

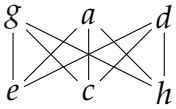


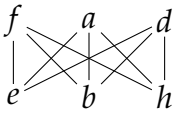
ou sous forme de matrice d'adjacence $M = \begin{bmatrix} 0&1&1&0&1&0&0&1\\ 1&0&0&1&0&1&0&0\\ 1&0&0&1&0&0&1&0\\ 0&1&1&0&1&0&0&1\\ 1&0&0&1&0&1&1&0\\ 0&1&0&0&1&0&0&1\\ 0&0&1&0&1&0&0&1\\ 1&0&0&1&0&1&1&0 \end{bmatrix}$.

Comme il est assez difficile de se représenter ce graphe en trois dimensions, ou de lire la matrice, O'Neill aimerait une représentation planaire. Justifiez, en utilisant soit un critère de planarité vu en cours, soit le théorème de Kuratowski, que ce graphe n'est *pas* planaire.

Réponse :

Ce graphe possède 8 sommets et 14 arêtes. Comme il ne possède aucun triangle, il devrait vérifier $14 = |E| \leq 2|V| - 4 = 12$, ce qui n'est pas le cas.

Une autre démonstration consiste à dire que ce graphe ne peut être planaire puisqu'il contient 

ou encore 

c'est-à-dire une instance de $K_{3,3}$, qui ne peut pas être planaire d'après de théorème de Kuratowski.

2. **(1 pt)** Une fois par an, on demande à O'Neill de tester les portails en empruntant toutes les connexions possibles. Justifiez que ce graphe ne possède pas de chemin eulérien, c'est-à-dire qu'O'Neill ne peut pas visiter toutes les arêtes en n'empruntant qu'une seule fois chacune.

Réponse :

Un graphe connexe possède un chemin eulérien si et seulement si le nombre de sommets de degré impair est 0 ou 2. Or ici il y a 4 sommets de degré impair : b, c, f et g.

3. **(0.5 pt)** Quel est le diamètre de ce graphe ?

Réponse :
3

4. **(0.5 pt)** Quel est le rayon de ce graphe ?

Réponse :
2

5. **(2 pts)** Pour se déplacer plus vite, on demande à Thomas Anderson (alias *Néo*) de modifier la matrice pour installer de nouvelles connexions entre les portails. On souhaite maintenant relier tous les portails qui étaient connectés par un chemin de deux connexions.

   Par exemple il était possible de passer de a à g en chaînant deux connexions, le nouveau graphe doit donc posséder une arête entre a et g en plus de celles qui existaient auparavant. À l'inverse, il n'est pas possible de relier c à f avec une ou deux connexions, le nouveau graphe ne devra donc pas posséder d'arête entre c et f.

   Quelles opérations matricielles permettent de définir la nouvelle matrice M' à partir de M ? Vous pouvez interpréter les éléments de M dans le semi-anneau des booléens $(\mathbb{B}, \vee, \wedge, 0, 1)$ et utiliser les opérations matricielles qui en découlent.

   Note : on ne vous demande pas de calculer M', juste de donner une formule qui lie M' à M.

Réponse :
M^2 représente tous les paires de sommets accessibles par deux arêtes. On a donc $M' = M + M^2$.
Dans cette formule, l'addition et le produit matriciel utilisent $\vee$ et $\wedge$ à la place des addition et multiplications scalaires (sinon on verrait des 2 entre deux sommets qui peuvent être connectés soit par une arête, soit par deux arêtes).

6. **(2 pts)** L'année suivante, on rappelle Néo pour lui faire à nouveau effectuer la même opération : ajouter des arêtes pour relier les sommets qui étaient connectés par deux arêtes dans le graphe associé à la matrice M'.

   Néo réalise alors que la matrice M'' nouvellement formée correspond à un graphe complet.

   De façon générale, sur un graphe connexe de diamètre D, combien de fois faudrait-il itérer cette opération sur la matrice d'adjacence pour obtenir un graphe complet ?

Réponse :
À l'itération 1, on a relié tous les sommets à une distance ≤ 2.
À l'itération 2, on a relié tous les sommets à une distance ≤ 4.
$\vdots$
À l'itération k, on a relié tous les sommets à une distance $\leq 2^k$.
Si le diamètre du graphe est D, on obtiendra une graphe complet après la plus petite itération k telle que $2^k \geq D$. Autrement dit, $k = \lceil \log_2 D \rceil$.

2 Coloration de graphe (10 points)

Le *nombre chromatique* d'un graphe est le nombre de couleurs minimum nécessaire pour colorier les sommets du graphe de façon à ce que deux sommets voisins ne partagent pas la même couleur.

1. **(1 pt)** Quel est le nombre chromatique du graphe de l'exercice 1 ?

Réponse :
2

2. **(1 pt)** En utilisant 4 sommets numérotés de 1 à 4, il est possible de construire 2^6 graphes différents. Combien ont pour nombre chromatique 4 ?

Réponse :

Un seul : le graphe complet d'ordre 4.
Car s'il manque une arête (x, y) on peut donner la même couleurs à x et y, et on aura besoin au pire de 2 couleurs supplémentaires pour les deux sommets restant. Donc si un graphe de 4 sommets n'est pas complet, son nombre chromatique est < 4.

L'algorithme qui suit colorie un graphe en s'assurant qu'un sommet n'a pas la couleur de ses voisins. Les états sont coloriés dans un ordre σ passé en argument, en utilisant la première couleur qui n'est pas utilisée par les voisins. Le résultat peut utiliser plus de couleurs que le nombre chromatique du graphe.

```
GREEDYCOLOR(G = (V, E), σ)
Entrée : un graphe G, un ordre sur les sommets σ (σ est une permutation de V)
Sortie : un tableau de couleurs C indicé par les sommets
 1  for each c ∈ {1, ..., |V|}      // marquer toutes les couleurs comme disponibles
 2        Avail[c] ← 1
 3  for each x ∈ V                  // initialement les sommets ne sont pas coloriés
 4        C[x] ← 0
 5  for each x in σ :               // itération sur les sommets dans l'ordre donné
 6        for each y ∈ succ(x) :    // repérage des couleurs voisines
 7              Avail[C[y]] ← 0
 8        i ← 1
 9        while Avail[i] = 0        // recherche de la première couleur libre
10              i ← i + 1
11        C[x] ← i                  // affectation de la couleur trouvée
12        for each y ∈ succ(x) :    // remise à disposition des couleurs voisines
13              Avail[C[y]] ← 1
14  return C
```

Dans cet algorithme, les couleurs sont désignées par des numéros de 1 à $|V|$. La valeur $C[x]$ donne la couleur du sommet x, ou 0 s'il n'est pas colorié. Le tableau *Avail* est utilisé pour repérer les couleurs utilisées par les sommets voisins, afin de pouvoir trouver la première couleur inutilisée (notez que $Avail[0]$ peut changer de valeur aux lignes 7 et 13, mais ne sera jamais lu, la boucle de ligne 9 commençant à l'indice 1).

Un suppose que le graphe G est non-orienté, et représenté à l'aide de listes d'adjacence.

3. **(2 pts)** Lors d'une exécution complète de l'algorithme, combien de fois la ligne 7 est-elle exécutée ? Donnez une réponse précise en fonction de $|V|$ et $|E|$.

Réponse :

$\sum_x \deg(x) = |E|$ fois.
Notez que dans un graphe non-orienté $E \subseteq V \times V$ contient deux fois chaque arête puisque l'adjacence est une relation symétrique ($(x, y) \in E \iff (y, x) \in E$).

4. **(2 pts)** Pour un x donné (c'est-à-dire pour une itération de la boucle de la ligne 5), quel est le nombre maximal d'exécutions de la ligne 10 ? Donnez une réponse précise en fonction de $|V|$, $|E|$, et $\deg(x)$.

Réponse :

Il ne peut y avoir plus de $\deg(x)$ valeurs à 1 dans le tableau *Avail*. Le test de la ligne 9 ne peut donc réussir que $\deg(x)$ fois au plus, et la ligne 10 sera donc exécutée au plus $\deg(x)$ fois.

5. **(1 pts)** En déduire une borne supérieure du nombre total d'exécutions de la ligne 10 (tous x

confondus). Donnez une réponse précise en fonction de $|V|$ et $|E|$.

Réponse :

La ligne 10 est exécutée au pire $\sum_x \deg(x) = |E|$ fois.

6. **(2 pts)** Quelle est la complexité de l'algorithme GREEDYCOLOR ? Donnez votre réponse en fonction de $|E|$ et $|V|$, et précisez bien s'il s'agit d'un $O(...)$ or un $\Theta(...)$.

Réponse :

$\Theta(|V| + |E|)$

7. **(1 pt)** Comment peut-on simplifier l'expression de cette complexité si l'on sait que le graphe est connexe ?

Réponse :

Si le graphe est connexe, on sait que $|V| = O(|E|)$ donc la complexité $\Theta(|V| + |E|)$ se simplifie en $\Theta(|E|)$.

3 Plus courts chemins (4 points)

On veut calculer les distances entre toutes les paires de sommets d'un graphe $G = (V, E, m)$ orienté, connexe, et pondéré par des longueurs strictements positives ($\forall (x, y) \in E$, $m(x, y) > 0$). Est-il plus intéressant d'utiliser l'algorithme de Floyd-Warshal, ou d'appliquer $|V|$ fois l'algorithme de Dijkstra ? Justifiez votre réponse à l'aide des complexités.

Réponse :

Floyd-Warshal coûte $\Theta(|V|^3)$.
Dijkstra coûte $O((|E| + |V|) \log |V|)$ ou $O(|E| + |V| \log |V|)$ selon le type de tas utilisé.
La première complexité est celle obtenue avec un tas binaire. Comme le graphe est connexe, on peut la simplifier en $O(|E| \log |V|)$. Maintenant s'il faut exécuter l'algorithme $|V|$ fois, cela nous donne $O(|V|\dot{|}E| \log |V|)$. Le choix entre répéter Dijkstra (avec tas binaire) ou Floyd-Warshal dépend donc de la densité du graphe : si $|E| \log |V|$ est plus petit que $|V|^2$, on préfèra Dijkstra,
Dans le cas de Dijkstra implémenté avec un tas de Fibonnaci, la réponse est beaucoup plus simple. Répéter Dijkstra coûte $O(|V|\dot{|}E| + |V|^2 \log |V|)$ ce qui est toujours inférieur à $\Theta(|V|^3)$. Répéter Dijkstra (avec tas de Fibonnaci) est donc plus intéressant qu'exécuter Floyd-Warshal.