Écrit Blanc d'informatique
Préparation au CAPES de Mathématiques
19 décembre 2018
Durée 4h

Instructions :

1. Les problèmes sont indépendants et *doivent* être traités sur des copies séparées.
2. La clarté et la précision des réponses font partie intégrante de l'évaluation.
3. Des graphes sans dessin ...

1 Problème 1 : Coloriage de Graphes

Problème proposé par Laure Gonnord

Préliminaires

Ce sujet est adapté du sujet X-ENS 2018, dont il reprend une partie de l'énoncé.

Complexité Par **complexité en temps** d'un algorithme A, on entend le nombre d'opérations élémentaires (comparaison, addition, soustraction, multiplication, division, affectation, test, etc) nécessaires à l'exécution de A dans le cas le pire.

Lorsque la complexité en temps ou en espace dépend d'un ou plusieurs paramètres $k_0, \ldots, k_{r-1}$, on dit que A a une complexité $O(f(k_0, \ldots, k_{r-1}))$ s'il existe une constante $C > 0$ telle que, pour toutes les valeurs de $k_0, \ldots, k_{r-1}$ suffisamment grandes (c'est à dire plus grandes qu'un certain seuil), pour toute instance du problème de paramètres $k_0, \ldots, k_{r-1}$, la complexité est au plus $Cf(k_0, \ldots, k_{r-1})$.

On dit que la complexité en temps est **linéaire** quand f est une fonction linéaire des paramètres $k_0, \ldots, k_{r-1}$, **polynomiale** quand f est une fonction polynomiale des paramètres $k_0, \ldots, k_{r-1}$ et enfin **exponentielle** quand $f = 2^g$ où g est une fonction polynomiale des paramètres $k_0, \ldots, k_{r-1}$.

Les complexités (en temps) des algorithmes **devront être justifiées**.

Graphes Rappelons qu'un graphe non-orienté est la donnée (S, A) de deux ensembles finis :

— un ensemble S de **sommets**, et

— un ensemble $A \subset S \times S$ d'**arêtes**, tel que pour tout couple de sommets (s, t), $s \neq t$ et, on a $(s, t) \in A$ si et seulement si $(t, s) \in A$.

Etant donné un graphe $G = (S, A)$, le **sous-graphe induit** par un ensemble de sommets $T \subset S$ est $(T, A \cap (T \times T))$.

Soit $G = (S, A)$ un graphe et soit $s \in S$ un sommet de G. Un **voisin** de s est un sommet t de G qui est relié à s par une arête, c'est à dire tel que $(s, t) \in A$. On note $V(s)$ l'ensemble des voisins de s. Le **degré** $d(s)$ de s est le cardinal de $V(s)$. Le **degré** $d(G)$ de G est le maximum des degrés de ses sommets.

Un graphe est dit **étiqueté** lorsque l'on dispose d'une fonction, dite d'étiquetage, de l'ensemble de ses sommets vers un ensemble non vide arbitraire, que l'on appelle ensemble des étiquettes. Les étiquettes peuvent par exemple être des entiers, des listes ou des chaînes de caractères.

On dit qu'une fonction d'étiquetage L est un **coloriage** des sommets de $G = (S, A)$ lorsque deux sommets voisins ont toujours deux étiquettes distinctes (alors appelées **couleurs**), c'est à dire lorsque L vérifie la condition

$$\forall s, t \in S,\ (s, t) \in A \Rightarrow L(s) \neq L(t)$$

Un graphe est dit k-coloriable s'il admet un coloriage avec au plus k couleurs. Un graphe est dit colorié s'il est k-coloriable pour un $k > 0$.

Le **nombre chromatique** d'un graphe non orienté G, noté $\chi(G)$, est le nombre minimal k tel que G est k-coloriable. Cet énoncé porte sur le calcul de nombres chromatiques et de coloriages.

Représentation des graphes étiquetés On se fixe dans cet énoncé une représentation des graphes par matrices d'adjacence (avec 0/1). On se fixe également comme convention que les étiquetages des graphes sont tous à valeurs entières. L'étiquetage d'un graphe sera donné par une liste d'entiers. Un graphe non orienté $G = (S, A)$ avec $S = \{0, \ldots, n-1\}$ est représenté par une valeur `gphe` de type `graphe` telle que pour $i, j \in S$, `gphe[i,j]=1` si et seulement si $(i, j) \in A$. Le graphe G étant supposé non orienté, on a alors également par symétrie `gphe[j,i]=1`. Pour un étiquetage `etiq` de `gphe`, l'étiquette du sommet i de `gphe` est donnée par `etig[i]`.

En Python/igraph, on crée une matrice d'adjacence comme ceci :

```
# ceci est un graphe avec 6 sommets
adj = np.array([[0, 1, 1, 0, 0, 0],
                [1, 0, 0, 1, 0, 0],
                [1, 0, 0, 0, 1, 0],
                [0, 1, 0, 0, 1, 0],
                [0, 0, 1, 1, 0, 1],
                [0, 0, 0, 0, 1, 0]])
```

et un étiquetage comme ceci :

```
etiq = [1, 1, 2, 2, 1, 0]
```

1.1 Coloriage

Question #1

Indiquer, pour chacun des graphes de la figure 1, si l'étiquetage proposé est un coloriage.

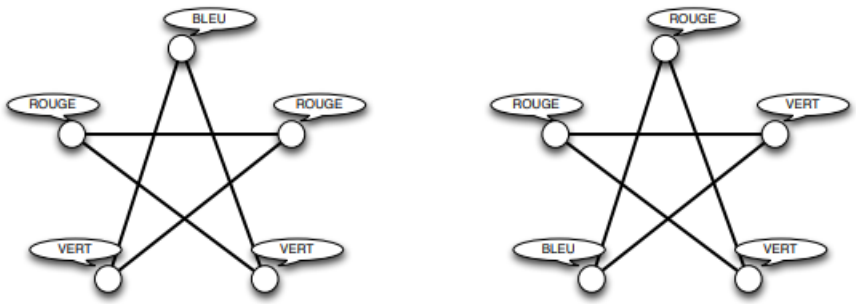

FIGURE 1 – Graphes (bien) coloriés ?

Question #2

Donner le nombre chromatique, ainsi qu'un exemple de coloriage pour le **graphe de Petersen** de la Figure 2.

La vérification de la propriété de coloriage est le problème suivant.
— Entrée : un graphe G et un étiquetage L de G.
— Question : L est-il un coloriage de G ?

Question #3

Ecrire une fonction `est_col`, telle que `est_col(gphe,etiq, taille)` renvoie `True` si et seulement si `etiq` est un coloriage de `gphe` (taille est le nombre de sommets).

Dans le cas où la taille de l'étiquetage est strictement inférieure au nombre de sommets du graphe, la fonction renvoie `False`. On demande une complexité quadratique en le nombre de sommets du graphe.

Question #4

Démontrer que le calcul du nombre chromatique d'un graphe peut s'effectuer en temps exponentiel en le nombre de sommets. *indication : on se demandera combien il existe de coloriages à k couleurs, pour k inférieur au nombre de sommets du graphes.*

1.2 2-coloriage

Nous avons vu à la question 4 que le calcul du nombre chromatique peut s'effectuer en temps exponentiel en le nombre de sommets du graphe. Dans le cas général, on ne sait aujourd'hui pas faire mieux. Pour obtenir de meilleures bornes de complexité, il faut donc se limiter à des sous-probèmes. On considère dans cette partie le cas du 2-coloriage.

Graphe biparti. Un graphe G est **biparti** lorsque l'ensemble de ses sommets S peut être divisé en deux sous-ensembles disjoints T et U (non vides), tels que chaque arête a une extrémité dans T et l'autre dans U.

Question #5

Démontrer (proprement) qu'un graphe G est biparti si et seulement s'il est 2-coloriable.

On se propose de programmer la vérification de la 2-colorabilité des graphes en procédant comme suit. On effectue un parcours du graphe en profondeur au cours duquel on construit une 2-coloration du graphe. On se donne pour ce faire trois étiquettes, disons -1, 0 et 1. L'étiquetage est initialisé à -1 pour tous les sommets, et on teste la 2-colorabilité avec 0 et 1. Le principe de l'algorithme est le suivant.

(1) On choisit un sommet s d'étiquette -1.
(2) On colorie les sommets rencontrés lors du parcours en profondeur à partir de s, en alternant entre les couleurs 0 et 1 à chaque incrémentation de la profondeur, et en vérifiant si les sommets déjà coloriés rencontrés sont d'une couleur compatible.
(3) Enfin, s'il reste des sommets d'étiquette -1, alors on revient au point (1).

Question #6

Écrire une fonction récursive `explo(gr, i, k, taille, etiq)` qui réalise le parcours en profondeur du graphe `gr` à partir du sommet `i`, en fixant la couleur de ce sommet à `k` dans `etiq`. Avec quelle couleur doivent être coloriés les voisins du sommet k ?

Comme les paramètres des fonctions python sont mutables, toute modification apportée à `etiq` sera (automatiquement) enregistrée à la sortie de la fonction. -1 *dans le tableau `etiq` dénote le fait qu'un sommet n'a pas encore été vu.*

Question #7

En utilisant la fonction précédente, écrire une fonction `deuxcol(gphe, taille)` qui calcule et retourne un 2 coloriage (0/1) du graphe si celui-ci est 2-coloriable. Le sommet 0 sera colorié en 0. Faire un exemple.

On demande une complexité quadratique en le nombre de sommets du graphe (justifier !). Le comportement de la fonction est laissé au choix du candidat lorsque le graphe n'est pas 2-coloriable.

Indication : l'initialisation des étiquettes peut être réalisée avec : `etiq = [-1] * taille`

1.3 Glouton

Dans cette partie, nous allons étudier un algorithme permettant de colorier un graphe en temps polynomial, mais donnant en général un coloriage sous-optimal : le coloriage obtenu peut dans certains cas utiliser plus de couleurs que le coloriage optimal.

Cet algorithme prend en paramètre un ordre sur les sommets du graphe, que l'on appelera **ordre de numérotation**.
Par exemple, $1 < 3 < 4 < 0 < 2 < 6 < 5 < 9 < 8 < 7$ et $0 < 7 < 2 < 5 < 4 < 6 < 8 < 1 < 3 < 9$ sont deux ordres de numérotation des sommets du graphe de Petersen (Figure 2).

Pour un graphe `gphe` à n sommets, on implémente un ordre de numérotation de ses sommets par un tableau `num` de n valeurs entières, tel que `num[k]`$=j$ si et seulement si le sommet j apparaît en $(k+1)$-ième position dans l'ordre.
L'**algorithme glouton** construit un coloriage L d'un graphe G en utilisant au plus $d(G)+1$ couleurs. Son principe est le suivant :
On parcourt la liste des sommets du graphe, dans l'ordre de numérotation des sommets donné. Pour chaque sommet s parcouru :
(1) On calcule l'ensemble $C(s) = \{L(t)/\ t \in V(s)\}$ des couleurs déjà données aux voisins de s.
(2) On cherche le plus petit entier naturel c qui n'appartient pas à $C(s)$.
(3) On pose $L(s) = c$.

Question #8

Considérons le graphe de Petersen (Figure 2) et les deux ordres de numérotation :

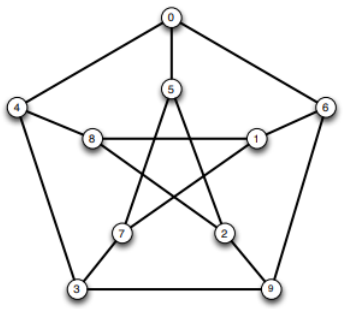

FIGURE 2 – Le graphe de Peterson à 10 sommets

```
num1 = [1, 3, 4, 0, 2, 6, 5, 9, 8, 7]
num2 = [0, 7, 2, 5, 4, 6, 8, 1, 3, 9]
```

Donner les coloriages obtenus par l'algorithme glouton décrit ci-dessus pour le graphe de Petersen et chacun de ces deux ordres de numérotation, ainsi que les nombres de couleurs correspondants.

Question #9

Écrire une fonction `min_couleur_possible(gr, etiquetage, taille, sommet)` qui pour un graphe `gphe` à *taille* sommets, un étiquetage `etiquetage` à valeurs dans $\{-1, \dots, n-1\}$, renvoie le plus petit entier naturel n'appartenant pas à l'ensemble $\{\texttt{etiq}[t] / \ t \in V(\texttt{s})\}$. On demande une complexité $O(n)$.

Question #10

Écrire une fonction `colore_glouton` : pour un graphe `gphe` de taille *taille* et un ordre de numérotation `num` de ses sommets, l'appel `colore_glouton(gphe, num, taille)` renvoie le coloriage glouton de `gphe` selon l'ordre, avec au plus $d+1$ sommets, où d est le degré de `gphe`. On demande une complexité $O(n^2)$, où n est le nombre de sommets de `gphe`.

Dans le cas où le tableau `num` contient autre chose qu'un ordre de numérotation des sommets de `gphe`, le résultat de la fonction est laissé au choix, mais il faudra préciser.

Question #11

Montrer que l'algorithme de coloriage glouton construit toujours un coloriage, et que ce coloriage utilise au plus $d+1$ couleurs, où d est le degré du graphe en entrée.

Question #12

Soit G un graphe. Montrer que pour tout coloriage L de G, il existe un ordre de numérotation des sommets tel que le coloriage glouton L' associé vérifie $L'(s) \leq L(s)$ pour tout sommet s de G. En déduire qu'il existe une numérotation des sommets telle que l'algorithme glouton renvoie un coloriage optimal.

Les questions 7 et 11 indiquent que l'efficacité de l'algorithme glouton est en grande partie dépendante de l'ordre dans lequel on choisit de parcourir les sommets du graphe. L'ordre correspondant à la représentation choisie du graphe (dans notre cas, les indices de la matrice d'adjacence, c'est à dire la permutation identité) est le plus simple à calculer, mais a peu de chances d'être efficace. A contrario, on pourrait essayer de déterminer l'ordre optimal, dont on a prouvé l'existence à la question 11, mais cela n'apporte aucun bénéfice vis-à-vis de la complexité temporelle du problème.

Une alternative est donnée par l'optimisation de Welsh-Powell. L'idée est de parcourir l'ensemble des sommets du graphe par ordre de degré décroissant. Le tri des sommets par degré décroissant ne prend pas plus de temps que le parcours glouton, mais permet d'obtenir un algorithme raisonnablement efficace en pratique.

Question #13

Écrire une fonction de tri **décroissant** d'un tableau / d'une liste d'entiers en Python, et donner sa complexité.

Question #14

Écrire une fonction `degres(gr, taille)` qui retourne la liste des degrés des sommets du graphe.

Question #15

En déduire une fonction `welsh_powell` qui implémente l'optimisation de Welsh-Powell. *Une modification des deux algorithmes précédents pourra être utile.* Pourquoi un tri quadratique est-il suffisant ?