ÉCOLE POLYTECHNIQUE — ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2018 **FILIÈRES MP SPECIALITÉ INFO**

COMPOSITION D'INFORMATIQUE – A – (XULCR)

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation sera **obligatoirement** Caml Light.

*
* *

Nombre chromatique et coloriage de graphe

Préliminaires

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 pire cas.

Lorsque la complexité en temps dépend d'un ou plusieurs paramètres $\kappa_1, \ldots, \kappa_r$, on dit qu'elle est en $O(f(\kappa_1, \ldots, \kappa_r))$ s'il existe une constante $C > 0$ telle que, pour toutes les valeurs de $\kappa_1, \ldots, \kappa_r$ suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), la complexité est au plus $C \times f(\kappa_1, \ldots, \kappa_r)$.

On dit que la complexité en temps est **linéaire** quand f est une fonction linéaire des paramètres $\kappa_1, \ldots, \kappa_r$, **polynomiale** quand f est une fonction polynomiale des paramètres $\kappa_1, \ldots, \kappa_r$, et enfin **exponentielle** quand $f = 2^g$, où g est une fonction polynomiale des paramètres $\kappa_1, \ldots, \kappa_r$.

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

Étant donné un graphe $G = (S, A)$, le **sous-graphe induit** par un ensemble de sommets $T \subseteq S$ 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.

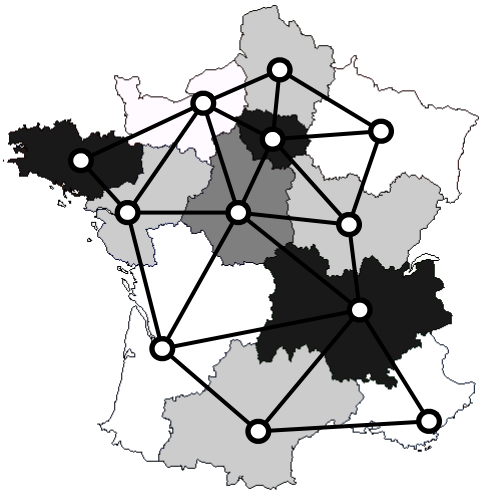


FIGURE 1 – Exemple de graphe colorié : le graphe des régions métropolitaines françaises (hors Corse). Deux régions sont reliées par une arête dans le graphe si et seulement si elles sont voisines. Deux régions voisines sont de couleurs différentes.

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 (1) suivante :

$$\forall s, t \in S, \quad (s,t) \in A \;\Rightarrow\; L(s) \neq L(t). \tag{1}$$

Un graphe G 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 $k > 0$. Un exemple de graphe colorié est donné sur la figure 1 ci-dessus.

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. On se fixe également comme convention que les étiquetages des graphes sont tous à valeurs entières. L'étiquetage d'un graphe sera donné par un tableau d'entiers. On se donne les types Caml Light suivants :

```
type graphe == bool vect vect;;
type etiquetage == int vect;;
```

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 tous sommets `i`, `j` $\in S$, on ait `gphe.(i).(j) = true` si et seulement si (`i`, `j`) $\in A$. Le graphe G étant supposé non-orienté, on a alors également par symétrie

`gphe.(j).(i) = true`. Pour un étiquetage `etiq` de `gphe`, l'étiquette du sommet `i` de `gphe` est donnée par `etiq.(i)`.

En plus des fonctionnalités de base du langage Caml Light, le candidat pourra utiliser les fonctions suivantes sans les programmer :

— `make_vect : int -> 'a -> 'a vect`
`make_vect n v` renvoie un vecteur de longueur `n` et dont toutes les cases valent `v`.

— `vect_length : 'a vect -> int`
`vect_length t` renvoie la longueur de `t`.

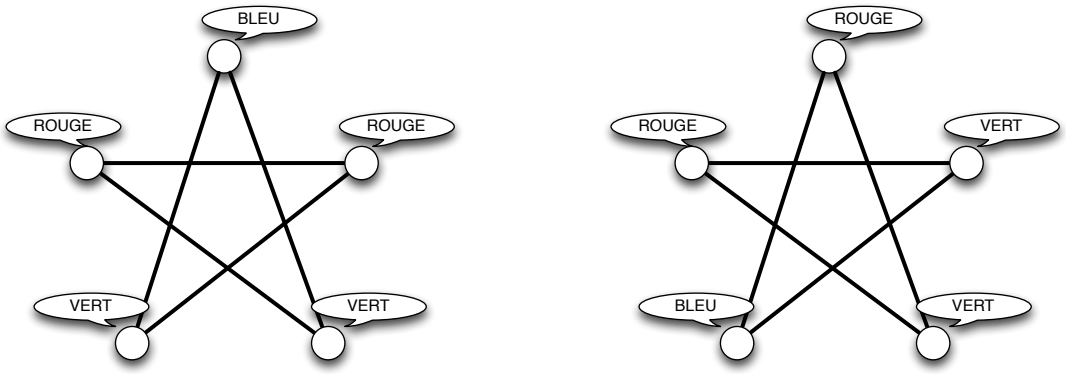
— `list_length : 'a list -> int`
`list_length l` renvoie la longueur de `l`.

— `vect_of_list : 'a list -> 'a vect`
`vect_of_list l` renvoie un vecteur `t` de même longueur que `l`, qui contient les mêmes éléments que `l` et dans le même ordre.

— `range : int -> int vect`
`range n` renvoie le vecteur `[|0,...,n-1|]`.

1 Coloriage

Question 1. Indiquer, pour chacun des graphes suivants, s'il est colorié :



Question 2. Donner le nombre chromatique, ainsi qu'un exemple de coloriage correspondant, pour le **graphe de Petersen** représenté figure 2 page 4.

Question 3. 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 ?

Écrire une fonction `est_col : graphe -> etiquetage -> bool`, telle que `est_col gphe etiq` renvoie `true` si et seulement si `etiq` est un coloriage de `gphe`. 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.

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

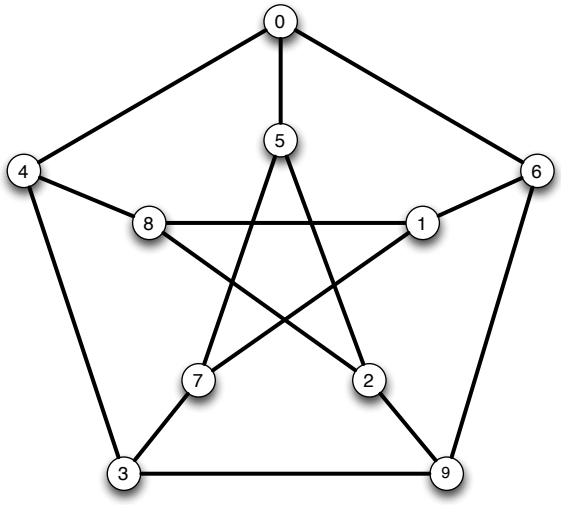


FIGURE 2 – Le graphe de Petersen, de sommets $0, \ldots, 9$.

pas faire mieux. Pour obtenir de meilleures bornes de complexité, il faut donc se limiter à des sous-problèmes. 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, tels que chaque arête a une extrémité dans T et l'autre dans U.

Question 5. Démontrer 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 `deux_col : graphe -> etiquetage` telle que `deux_col gphe` renvoie un 2-coloriage de `gphe` si `gphe` est 2-coloriable. Le coloriage utilisera les couleurs `0` et `1`. On demande une complexité quadratique en le nombre de sommets du graphe. Le comportement de la fonction est laissé au choix du candidat lorsque `gphe` n'est pas 2-coloriable.

Indication : on pourra se donner un étiquetage `etiq : etiquetage` *de longueur* `(vect_length gphe)`*, dont toutes les cases sont initialisés à* `-1`*, et que l'on met à jour au fur et à mesure du parcours de* `gphe`.

3 Algorithmes gloutons

Dans cette partie, nous allons étudier deux algorithmes 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.

Ces deux algorithmes prennent 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 page 4).

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 $(\texttt{k}+1)^{\text{ième}}$ position dans l'ordre.

Nous commençons par l'**algorithme glouton** de coloriage. Cet algorithme 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 $C(s) = \{L(t) \mid t \in V(s)\}$ 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 7. Considérons le graphe de Petersen (figure 2, page 4) et les deux ordres de numérotation `num1` $= [|1;3;4;0;2;6;5;9;8;7|]$ et `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 8. Écrire une fonction `min_couleur_possible : graphe -> etiquetage -> int -> int` telle que pour un graphe `gphe` à n sommets, un étiquetage `etiq` à valeurs dans $\{-1, \ldots, n-1\}$, et pour un sommet `s` de `gphe`, l'appel de `min_couleur_possible gphe etiq s` renvoie le plus petit entier naturel n'appartenant pas à l'ensemble $\{\texttt{etiq.(t)} \mid \texttt{t} \in V(\texttt{s})\}$. On demande une complexité en $O(n)$.

Question 9. Écrire une fonction `glouton : graphe -> int vect -> etiquetage`, telle que, pour un graphe `gphe` et un ordre de numérotation `num` de ses sommets, `glouton gphe num` renvoie le coloriage glouton de `gphe`, avec au plus $d+1$ couleurs, où d est le degré de `gphe`. On demande une complexité en $O(n^2)$, où n est le nombre de sommets de `gphe`.

Dans le cas où le tableau `num` contient autre chose qu'un ordre de numération des sommets de `gphe`, le résultat de la fonction est laissé au choix du candidat.

Question 10. 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 11. 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'apporterait 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 12. Écrire une fonction `tri_degre: graphe -> int vect`, qui calcule le tableau des sommets d'un graphe trié par ordre décroissant de leurs degrés. En déduire une fonction `welsh_powell: graphe -> etiquetage` qui implémente l'optimisation de Welsh-Powell, et justifier le choix de votre algorithme de tri pour la fonction `tri_degre`.

4 Algorithme de Wigderson

Considérons un graphe G avec n sommets. Supposons que G soit 3-coloriable, mais que l'on ait cette information sans pour autant effectivement disposer d'un 3-coloriage de G. Trouver un 3-coloriage de G pourrait prendre un temps exponentiel en n.

L'algorithme de Wigderson permet, pour un graphe G supposé 3-coloriable, de trouver en temps polynomial en n un coloriage de G avec $O(\sqrt{n})$ couleurs (au sens où il existe $C > 0$ tel que pour tout n suffisamment grand, ce coloriage ait au plus $C\sqrt{n}$ couleurs).

Cet algorithme repose sur la propriété établie dans la question qui suit.

Question 13. Soit $k > 0$. Montrer que si G est $(k+1)$-coloriable, alors pour tout sommet s de G le sous-graphe induit par $V(s)$ est k-coloriable.

Voici le principe de l'algorithme de Wigderson. Soit G un graphe à n sommets, et tel que G est 3-coloriable.

(1) On se donne comme couleur initiale $c = 0$.

(2) Pour chaque sommet s de G pas encore colorié et ayant au moins $\sqrt{n}$ voisins pas encore coloriés :

   (a) On 2-colorie, avec les couleurs c et $c+1$, le sous-graphe induit par l'ensemble des voisins de s pas encore coloriés.

   (b) On incrémente c du nombre de couleurs utilisées en (a).

(3) Enfin, on utilise l'algorithme glouton (avec un ordre de numérotation quelconque) pour colorier, avec des couleurs supérieures ou égales à c, le sous-graphe induit par l'ensemble des sommets pas encore coloriés.

Question 14. Montrer que l'algorithme de Wigderson appliqué à un graphe 3-coloriable construit toujours un coloriage, et que ce coloriage utilise un nombre de couleur en $O(\sqrt{n})$, où n est le nombre du sommets du graphe.

Nous allons maintenant implémenter cet algorithme. Commençons par programmer quelques fonctions auxiliaires simples.

Question 15. Écrire une fonction `sous_graphe : graphe -> int vect -> graphe` telle que pour `gphe : graphe` et `sg : int vect`, si `sg` est à valeurs dans {`0`, ..., `(vect_length gphe)-1`} et sans répétition, alors `sous_graphe gphe sg` renvoie la matrice d'adjacence du graphe de

sommets $\{0, \ldots,$ `(vect_length sg)-1`$\}$, et qui a une arête entre les sommets `s` et `t` si et seulement si `gphe.(sg.(s)).(sg.(t))` = `true`.

Dans le cas où `sg` a des valeurs hors de $\{0, \ldots,$ `(vect_length gphe)-1`$\}$, ou a des répétitions, le comportement de la fonction `sous_graphe` est laissé au choix du candidat.

Nous nous proposons d'utiliser pour les étiquetages la même convention que précédemment : on se donnera un étiquetage `etiq : etiquetage` de longueur `(vect_length gphe)`, initialisé à `-1`, et que l'on mettra à jour au fur et à mesure de l'algorithme.

Question 16. Écrire une fonction `voisins_non_colories : graphe -> etiquetage -> int -> int list` telle que `voisins_non_colories gphe etiq s` renvoie la liste des voisins `t` de `s` tels que `etiq.(t)` = `-1`.

En déduire une fonction `degre_non_colories : graphe -> etiquetage -> int -> int` telle que `degre_non_colories gphe etiq s` renvoie le nombre de voisins `t` de `s` tels que `etiq.(t)` = `-1`.

Question 17. Écrire une fonction `non_colories : graphe -> etiquetage -> int list` telle que `non_colories gphe etiq` renvoie la liste des sommets `s` de `gphe` tels que `etiq.(s)` = `-1`.

Nous disposons maintenant de toutes les briques nécessaires à l'implémentation de l'algorithme de Wigderson.

Question 18. Écrire une fonction `wigderson : graphe -> etiquetage` telle que si `gphe` est 3-coloriable, alors `wigderson gphe` renvoie un coloriage de `gphe` obtenu par l'algorithme de Wigderson décrit plus haut. On demande une complexité polynomiale en le nombre de sommets de `gphe`. De plus, les propriétés sur le coloriage établies à la question 14 devront être respectées et justifiées.

Le comportement de la fonction est laissé au choix du candidat lorsque `gphe` n'est pas 3-coloriable.

Question 19. Comment pourrait-on étendre l'algorithme de Wigderson à des graphes de nombre chromatique connu et strictement supérieur à 3 ?

* *
*