COMPOSITION D'INFORMATIQUE n°4

(Durée : 4 heures)

L'utilisation de la calculatrice **n'est pas autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Coloration de graphe

La coloration de graphe est un problème difficile qui a de nombreuses applications comme l'ordonnancement, l'allocation de registres en compilation ou même la géographie. Le célèbre théorème des quatre couleurs, qui affirme qu'un graphe planaire peut être coloré avec quatre couleurs, est l'un des premiers grands résultats mathématiques prouvés à l'aide de l'outil informatique. Il permet notamment d'affirmer que les régions représentées sur une carte peuvent être colorées avec seulement quatre couleurs sans que deux régions voisines n'aient la même couleur.

En C, on supposera que les bibliothèques `stdlib.h` et `stdbool.h` ont été chargées.

Lorsque le candidat écrira une fonction, il pourra faire appel à des fonctions définies dans les questions précédentes, même si elles n'ont pas été traitées. Il pourra également définir des fonctions auxiliaires, mais devra préciser leurs rôles ainsi que les types et significations de leurs arguments. Les candidats sont encouragés à expliquer les choix d'implémentation de leurs fonctions lorsque ceux-ci ne découlent pas directement des spécifications de l'énoncé. Si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction de tester si les hypothèses sont bien satisfaites.

On identifiera une même grandeur écrite dans deux polices de caractères différentes, en italique du point de vue mathématique (par exemple n) et en Computer Modern à chasse fixe du point de vue informatique (par exemple `n`).

Sans précision supplémentaire, lorsqu'une question demande la complexité d'une fonction, il s'agira de la complexité temporelle dans le pire des cas. La complexité sera exprimée sous la forme $\mathcal{O}(f(n,m))$ où n et m sont les tailles des arguments de la fonction, et f une expression la plus simple possible. Les calculs de complexité seront justifiés succinctement.

1 Préliminaires

Dans l'ensemble du sujet, on considère des graphes non orientés non pondérés. On notera $G = (S, A)$ un graphe et $n = |S|$ sans systématiquement rappeler ces notations s'il n'y a pas d'ambiguïté.

Pour un graphe $G = (S, A)$, une *numérotation* de G est une fonction $c : S \to \mathbb{N}$. Pour c une numérotation de G et $s \in S$, on appelle *couleur de* s la valeur $c(s)$. Une *coloration* de G est une numérotation de G telle que deux sommets adjacents n'ont jamais la même couleur. Pour un entier $k \in \mathbb{N}^*$, une *k-coloration* de G est une coloration de G à valeurs dans $[\![0, k-1]\!]$, c'est-à-dire une fonction $c : S \to \{0, \ldots, k-1\}$ telle que pour toute arête $\{s, t\} \in A$, $c(s) \neq c(t)$.

La figure 1 donne deux numérotations possibles d'un même graphe G_0 :

La numérotation de gauche n'est pas une coloration car les sommets grisés (1 et 4) sont adjacents et ont la même couleur. La numérotation de droite est une 4-coloration. Attention, dans les représentations de la figure 1, les nombres à l'extérieur des sommets ne correspondent pas aux indices des sommets mais aux couleurs. Les indices des sommets sont à l'intérieur.

G est dit *k-colorable* s'il possède une k-coloration. On définit également le *nombre chromatique* de G, noté $\chi(G)$, comme étant le plus petit entier k tel que G est k-colorable.

FIGURE 1 – Deux numérotations de G_0.

Question 1 Déterminer le nombre chromatique du graphe G_0 représenté ci-dessus. On exhibera une coloration optimale, et on justifiera rigoureusement qu'il est impossible d'utiliser moins de numéros différents.

Question 2 Faire de même pour le graphe G_1 de la figure 2.

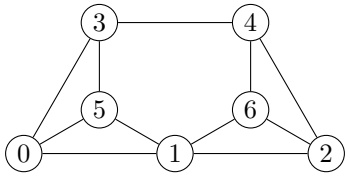


FIGURE 2 – Le graphe G_1.

Pour n un entier naturel supérieur ou égal à 3, on note $\mathcal{C}_n$ le graphe cyclique d'ordre n, c'est-à-dire $\mathcal{C}_n = (S, A)$ où $S = \{0, \ldots, n-1\}$ et $A = \{\{0, n-1\}\} \cup \{\{i, i+1\} \mid i \in [\![0, n-2]\!]\}$.

Question 3 Déterminer en justifiant le nombre chromatique de $\mathcal{C}_n$ en fonction de n.

Question 4 Déterminer en justifiant le nombre chromatique de $\mathcal{K}_n$, le graphe complet à n sommets en fonction de n.

Une *clique* de G est un ensemble de sommets deux à deux adjacents. Le *nombre clique* de G, noté $\omega(G)$, est la taille maximale d'une clique de G.

Question 5 Déterminer en justifiant une inégalité entre $\omega(G)$ et $\chi(G)$. Pour tout $n \geqslant 5$, décrire un graphe connexe à n sommets pour lequel cette inégalité est en fait une égalité et un graphe connexe à n sommets pour lequel cette inégalité est stricte.

Un *stable* de G est un ensemble de sommets deux à deux non adjacents. Le *nombre stable* de G, noté $\alpha(G)$, est la taille maximale d'un stable de G.

Question 6 Montrer que $\frac{n}{\alpha(G)} \leqslant \chi(G)$. Pour tout $n \geqslant 5$, décrire un graphe connexe à n sommets pour lequel cette inégalité est en fait une égalité et un graphe connexe à n sommets pour lequel cette inégalité est stricte.

2 Calcul du nombre chromatique

Dans cette partie, on cherche à calculer la valeur exacte du nombre chromatique d'un graphe. Pour ce faire, on va naïvement tester toutes les numérotations possibles, ne considérer que celles qui sont des colorations, en conservant le plus petit nombre de couleurs utilisées.

On représente en C une liste chaînée d'entiers par le type suivant :

```
struct Liste {
    int val;
    struct Liste* suivant;
};

typedef struct Liste liste;
```

Question 7 Écrire une fonction `void liberer_liste(liste* lst)` qui libère l'espace mémoire occupé par une liste.

Question 8 Écrire une fonction `liste* cons(int x, liste* lst)` qui prend en argument un entier `x` et une liste `lst` et renvoie la liste chaînée où on a rajouté un maillon de valeur `x` en tête de la liste `lst`.

Dès lors, un graphe G sera représenté par un tableau `G` de listes d'adjacence, c'est-à-dire un objet de type `liste**` de telle sorte que si $s \in S$, alors `G[s]` est la liste chaînée des voisins de s.

On représentera une numérotation $c : S \to \{0, \dots, k-1\}$ par un tableau `int*` de taille n contenant des éléments de $\{0, \dots, k-1\}$, telle que `c[s]` prend la valeur $c(s)$, pour tout sommet $s \in S$.

Question 9 Écrire une fonction `bool coloration(liste** G, int* c, int n)` qui prend en arguments un graphe `G`, une numérotation `c` et un entier n et renvoie `true` si `c` est une coloration de `G` et `false` sinon. On supposera que `G` et `c` sont de mêmes tailles n.

Pour la suite, on souhaite énumérer toutes les numérotations possibles d'un graphe à n sommets, utilisant $k \leqslant n$ couleurs distinctes, c'est-à-dire les k^n tableaux de taille n contenant des éléments de $\{0, \dots, k-1\}$. L'ordre choisi pour cette énumération est l'ordre lexicographique.

Question 10 Écrire une fonction `bool incrementer(int* c, int n, int k)` qui prend en arguments une numérotation `c` de taille `n` et un entier `k` et :

- modifie `c` en la numérotation suivante selon l'ordre lexicographique de $\{0, \dots, k-1\}^n$;
- renvoie `true` si (avant modification) `c` est un tableau ne contenant que la valeur $k-1$ (c'est-à-dire si on a atteint la dernière numérotation) et `false` sinon.

Si `c` ne contient que la valeur $k-1$, on modifiera `c` pour qu'il ne contienne que la valeur 0. Cette fonction devra avoir une complexité en $\mathcal{O}(n)$. mais on ne demande pas de le justifier

Par exemple, pour $n = 5$ et $k = 3$, si `c` contient $[2, 1, 0, 2, 2]$, alors `incrementer(c, n, k)` modifie `c` en $[2, 1, 1, 0, 0]$ et renvoie `false`. Si `c` contient $[2, 2, 2, 2, 2]$, alors `incrementer(c, n, k)` modifie `c` en $[0, 0, 0, 0, 0]$ et renvoie `true`.

Question 11 Montrer que, si ce n'est pas déjà le cas, on peut modifier la fonction précédente pour qu'elle ait une complexité amortie en $\mathcal{O}(1)$. On ne demande pas de coder cette éventuelle modification mais on attend une justification rigoureuse.

Question 12 En déduire une fonction `bool k_colorable(liste** G, int n, int k)` qui prend en arguments un graphe `G` d'ordre `n` et un entier `k` et renvoie `true` si G est k-colorable et `false` sinon.

Question 13 En déduire une fonction `int chi(liste** G, int n)` qui calcule et renvoie $\chi(G)$.

Question 14 Déterminer les complexités temporelles et spatiales dans le pire des cas de la fonction `chi` en fonction de n, le nombre de sommets de G. Cette fonction est-elle utilisable en pratique pour des graphes de taille 50 ? de taille 20 ?

3 NP-complétude

Dans cette partie, on cherche à montrer que le problème général du calcul du nombre chromatique d'un graphe est difficile. Pour ce faire, on étudie plutôt des problèmes de décision liés à la coloration. On définit les problèmes de décision suivants :

- Problème k-coloration (pour k fixé)
  - Entrée : un graphe G.
  - Question : le graphe G est-il k-colorable ?
- Problème Coloration
  - Entrée : un graphe G et un entier k.
  - Question : le graphe G est-il k-colorable ?

Par ailleurs, pour A et B deux problèmes de décision, la notation $A \leqslant_{\mathsf{m}}^{\mathsf{p}} B$ signifie qu'il existe une réduction many-one polynomiale de A à B.

Question 15 Montrer que pour tout entier k, k-coloration $\leqslant_{\mathsf{m}}^{\mathsf{p}}$ Coloration.

Question 16 Montrer que pour $h \leqslant k$ deux entiers fixés, h-coloration $\leqslant_{\mathsf{m}}^{\mathsf{p}}$ k-coloration.

Question 17 Montrer que 2-coloration $\in$ P. On décrira en français un algorithme polynomial permettant de résoudre le problème.

Question 18 Écrire une fonction `bool deux_colorable(liste** G, int n)` qui prend en argument un graphe `G` et renvoie `true` si G est 2-colorable et `false` sinon. Cette fonction devra avoir une complexité linéaire en le nombre de sommets et d'arêtes de G.

Les deux questions précédentes montrent que les problèmes 1-coloration et 2-coloration sont « faciles ». Les questions suivantes montrent la NP-complétude de 3-coloration.

Question 19 Montrer que Coloration $\in$ NP.

On rappelle les définitions suivantes : pour un ensemble de variables $V = \{v_1, \ldots, v_n\}$, un *littéral* est une variable ou sa négation, c'est-à-dire de la forme v_i ou $\overline{v_i}$. Une *clause* est une disjonction de littéraux (séparés par des $\vee$). Une formule est dite en *forme normale conjonctive* si c'est une conjonction de clauses (séparées par des $\wedge$). Elle est dite en 3-FNC si de plus chaque clause contient exactement 3 littéraux. Par exemple, la formule φ_0 suivante est en 3-FNC :

$$\varphi_0 = (a \vee b \vee c) \wedge \overbrace{(\overline{a} \vee c \vee \overline{d})}^{\text{clause}} \wedge (b \vee \underbrace{\overline{c}}_{\text{littéral}} \vee d)$$

On définit alors le problème de décision :

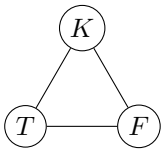
- Problème 3-SAT
  - Entrée : une formule booléenne φ en 3-FNC.
  - Question : la formule φ est-elle satisfiable ?

On admet que ce problème est NP-complet. La suite de cette partie a pour objectif de montrer qu'il se réduit en temps polynomial au problème 3-coloration. Pour ce faire, en considérant une formule booléenne φ en 3-FNC, on construit un graphe G_φ qui est 3-colorable si et seulement si φ est satisfiable. L'idée est que deux des couleurs utilisées correspondent à l'affectation *Vrai* ou *Faux* pour chaque littéral, et la troisième couleur est une couleur de contrôle.

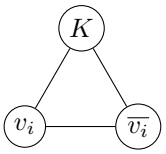
Dans le détail, si φ est une formule en 3-FNC sur un ensemble $V = \{v_1, \ldots, v_n\}$ de n variables, alors on définit le graphe $G_\varphi = (S, A)$ par :

- S contient trois sommets T (*True*), F (*False*) et K (*Control*), un sommet par littéral possible, et 5 sommets supplémentaires par clause ;

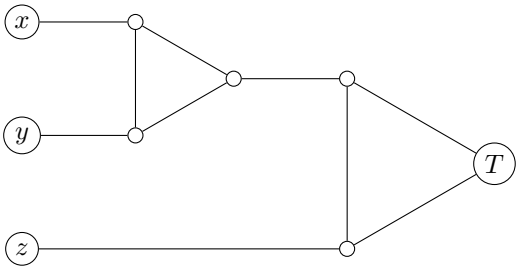
– A contient les arêtes suivantes :
  * $\{T, F\}, \{T, K\}, \{F, K\}$



  * pour chaque variable v_i, les arêtes $\{v_i, \overline{v_i}\}, \{v_i, K\}, \{\overline{v_i}, K\}$



  * pour chaque clause $C = x \vee y \vee z$ apparaissant dans φ (où x, y, z sont des littéraux), on rajoute les 10 arêtes telles que décrites dans le graphe ci-dessous, en utilisant les 5 sommets supplémentaires associés à la clause.



Par exemple, la figure 3 représente le graphe G_{φ_0} obtenu pour la formule booléenne φ_0 sur l'ensemble de variables $\{a, b, c, d\}$.

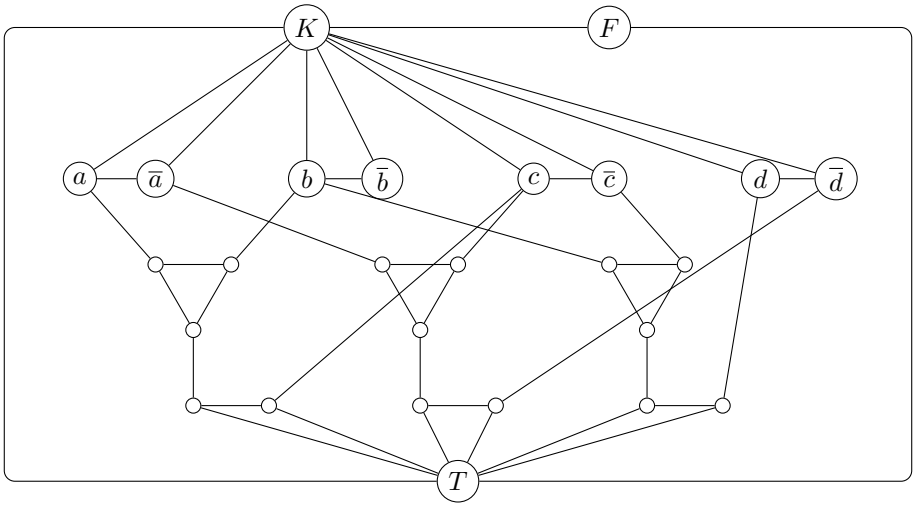


FIGURE 3 – Le graphe G_{φ_0} pour $\varphi_0 = (a \vee b \vee c) \wedge (\overline{a} \vee c \vee \overline{d}) \wedge (b \vee \overline{c} \vee d)$.

Question 20 Représenter graphiquement le graphe G_{φ_1} pour $\varphi_1 = (\overline{a} \vee b \vee \overline{c}) \wedge (a \vee b \vee \overline{c}) \wedge (a \vee \overline{b} \vee c)$.

Question 21 Justifier que la construction de G_φ à partir d'une formule φ en 3-FNC est en temps polynomial en fonction de la taille de φ (la taille d'une formule en 3-FNC est son nombre de clauses).

Question 22 Montrer que le graphe de la figure 4 est 3-colorable et que pour toute 3-coloration, au moins l'un des sommets x, y ou z a la même couleur que T.

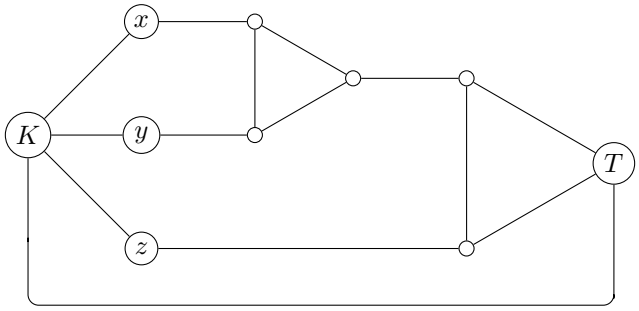


FIGURE 4 – Un gadget de clause.

Question 23 On suppose que φ est satisfiable. Montrer que G_φ est 3-colorable. On expliquera comment construire une 3-coloration de G_φ à partir d'une valuation de φ.

Question 24 On suppose que G_φ est 3-colorable. Montrer que φ est satisfiable. On expliquera comment construire une valuation de φ à partir d'une coloration de G_φ.

Question 25 En déduire que 3-coloration est NP-difficile, puis qu'il est NP-complet.

4 Algorithmes gloutons

Comme vu précédemment, le calcul du nombre chromatique d'un graphe est un problème difficile. On se propose donc d'étudier dans cette partie des algorithmes gloutons efficaces permettant de déterminer des k-colorations d'un graphe G, sans garantir que $k = \chi(G)$. Le schéma général de tous ces algorithmes est le même, seul l'ordre de traitement des sommets pouvant différer.

Début algorithme
  Entrée : graphe $G = (S, A)$
  Pour chaque sommet $s \in S$ non coloré **Faire**
    Colorer s en utilisant la plus petite couleur possible

La ligne « Colorer s en utilisant la plus petite couleur possible » consiste à parcourir tous les sommets colorés adjacents à s et à numéroter s en utilisant le plus petit numéro n'apparaissant pas parmi les couleurs des voisins de s. On attribue toujours la couleur 0 au premier sommet traité.

Question 26 Écrire une fonction `int plus_petit_absent(liste* lst)` qui prend en argument une liste `lst` d'entiers et renvoie le plus petit entier naturel n'apparaissant pas dans `lst`. Cette fonction devra avoir une complexité linéaire en la taille de `lst` dans tous les cas et on demande de justifier cette complexité.

Question 27 Écrire une fonction `liste* couleurs_voisins(liste** G, int* c, int s)` qui prend en argument un graphe `G`, une numérotation `c` et un sommet $s \in S$ et renvoie la liste chaînée des couleurs des voisins de s.

L'algorithme glouton le plus élémentaire consiste à parcourir les sommets dans l'ordre croissant des indices.

Question 28 Déterminer sans justifier la coloration renvoyée par cet algorithme sur le graphe G_1 de la figure 2.

Question 29 Écrire une fonction `int* colo_glouton(liste** G, int n)` qui prend en argument un graphe `G` d'ordre `n` et renvoie une coloration de G en suivant l'algorithme glouton.

Question 30 Déterminer la complexité temporelle de `colo_glouton`.

Dans un graphe $G = (S, A)$, le *degré* d'un sommet $s \in S$ est le nombre de sommets adjacents à s. On note $\Delta(G)$ le degré maximal d'un sommet de G.

Question 31 En s'appuyant sur l'algorithme glouton, donner en justifiant une majoration de $\chi(G)$ en fonction de $\Delta(G)$. Pour tout $n \geqslant 5$, décrire un graphe connexe à n sommets pour lequel cette inégalité est en fait une égalité et un graphe connexe à n sommets pour lequel cette inégalité est stricte.

Question 32 Déterminer une indexation des sommets telle que pour le graphe G_2 de la figure 5, la fonction `colo_glouton` ne renvoie pas une coloration optimale (c'est-à-dire utilisant strictement plus de $\chi(G_2)$ couleurs).

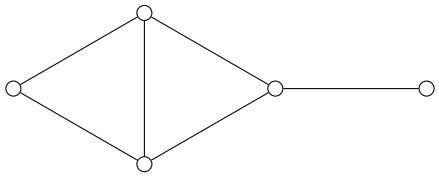


FIGURE 5 – Le graphe G_2.

Question 33 Montrer que pour tout graphe G, il existe une indexation des sommets telle que la fonction `colo_glouton` renvoie une coloration optimale.

L'algorithme de Welsh-Powell consiste à traiter les sommets par ordre décroissant de degré. En cas d'égalité de degrés, l'algorithme choisit le sommet de plus petit indice.

Question 34 Montrer que l'algorithme de Welsh-Powell renvoie toujours une coloration optimale pour le graphe G_2 de la figure 5, quelle que soit l'indexation des sommets.

Pour n un entier supérieur ou égal à 3, on appelle *graphe couronne à $2n$ sommets*, noté J_n, le graphe biparti $J_n = (S, A)$ défini par :

- $S = X \sqcup Y$ où $X = \{x_1, \ldots, x_n\}$ et $Y = \{y_1, \ldots, y_n\}$;
- $A = \{\{x_i, y_j\}, i \neq j\}$, c'est-à-dire qu'il existe une arête entre un sommet x_i et un sommet y_j si i et j sont distincts.

Les figures 6 et 7 montrent deux représentations possibles des graphes couronnes à 6, 8 et 10 sommets.

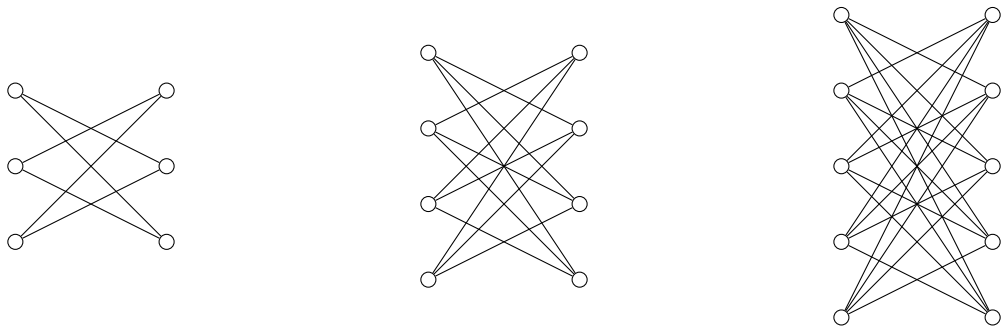


FIGURE 6 – Représentation de J_3, J_4 et J_5 de manière bipartie.

Question 35 Montrer que pour n supérieur ou égal à 3, il existe une indexation des sommets de J_n telle que l'algorithme de Welsh-Powell renvoie une 2-coloration, et une indexation telle que l'algorithme de Welsh-Powell renvoie une n-coloration.

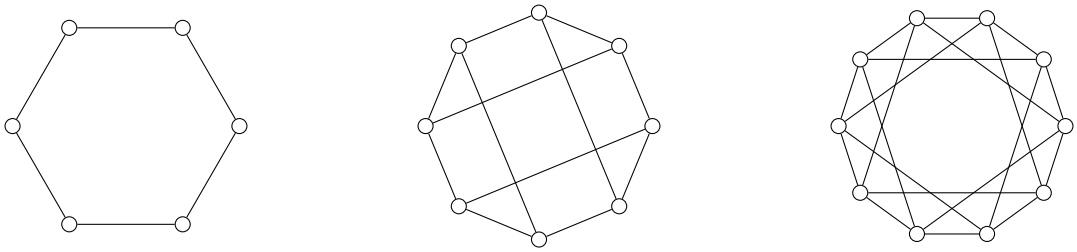


FIGURE 7 – Représentation de J_3, J_4 et J_5 de manière « couronne ».

Question 36 Montrer que quelle que soit l'indexation des sommets du graphe G_3 de la figure 8, l'algorithme de Welsh-Powell renvoie une coloration non optimale.

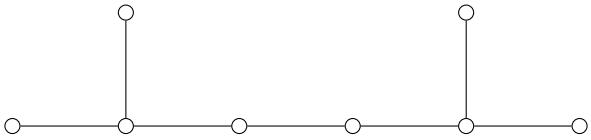


FIGURE 8 – Le graphe G_3.

⋆ ⋆ ⋆