



Samedi 6 avril 2024

**OPTION : SCIENCES DU NUMÉRIQUE**  
*MPI*

**DURÉE : 2 HEURES**

**Conditions particulières :**

Calculatrice interdite

Indiquez votre code candidat SCEI sur le QCM  
à insérer dans votre copie d'examen

# Concours CPGE EPITA-IPSA-ESME 2024

## Épreuve de Sciences du numérique MPI

Le sujet est composé d'un problème et d'un questionnaire à choix multiples. Le questionnaire à choix multiples devra être inséré dans votre copie. Les fonctions à produire dans ce sujet devront être rédigées dans le langage spécifié dans l'intitulé des parties et se conformer aux restrictions décrites ci-après. Il est possible d'écrire des fonctions auxiliaires non explicitement demandées à condition de les documenter et de les définir avant d'en faire usage.

### Restrictions des langages C et OCaml

Conformément au programme de l'enseignement d'informatique en classe de MPI, sont autorisés les traits et éléments techniques à connaître des langages C et OCaml. En particulier, l'usage des fonctions et modules OCaml ci-après sera considéré comme nul :

<code>Array.exists</code>	<code>List.exists</code>	<code>module Hashtbl</code>
<code>Array.for_all</code>	<code>List.filter</code>	<code>module Queue</code>
<code>Array.init</code>	<code>List.fold_left</code>	<code>module Stack</code>
<code>Array.iter</code>	<code>List.for_all</code>	
<code>Array.make_matrix</code>	<code>List.iter</code>	
<code>Array.map</code>	<code>List.map</code>	
<code>Array.mem</code>	<code>List.mem</code>	

### Problème – Coloration de graphes

On considère dans ce problème des **graphes non orientés**  $G = (V, E)$ , à  $n \in \mathbb{N}^*$  sommets, où  $V = \llbracket 0, n - 1 \rrbracket$  est l'ensemble des sommets de  $G$  et  $E$  l'ensemble de ses arêtes.  $E$  est un ensemble de parties à 2 éléments de  $V$ .

On dit que  $c : V \rightarrow \mathbb{N}$  est une **coloration** des sommets de  $G$  lorsque deux sommets voisins  $v_1$  et  $v_2$  de  $V$  ont toujours des **couleurs**  $c(v_1)$  et  $c(v_2)$  distinctes. Formellement :  $\forall \{v_1, v_2\} \in E, c(v_1) \neq c(v_2)$ .

Dans le cas où une coloration de  $G$  existe, on dira que  $G$  est **colorable**.

De plus lorsqu'il existe  $k \in \mathbb{N}^*$  tel que  $c(V) \subseteq \llbracket 0, k - 1 \rrbracket$ , on dira que  $c$  est une  **$k$ -coloration** de  $G$  et que le graphe est  **$k$ -colorable**.

Une coloration d'un graphe  $G$  sera dite **optimale** ou **minimale** lorsqu'il n'existera pas de coloration de  $G$  avec moins de couleurs.

On appelle **graphe induit** de  $G = (V, E)$  par un sous-ensemble  $V'$  de ses sommets, le graphe  $G' = (V', E')$  où  $E'$  est l'ensemble des arêtes de  $E$  ayant leurs extrémités dans  $V'$ . Formellement  $E' = \{\{v_1, v_2\} \in E \mid (v_1, v_2) \in V' \times V'\}$ .

On appelle **stable** d'un graphe  $G = (V, E)$ , tout sous-ensemble non vide  $V'$  de sommets de  $G$  vérifiant :

$$\forall (v_1, v_2) \in V' \times V', \{v_1, v_2\} \notin E$$

Autrement dit,  $V'$  est un stable de  $G$  si et seulement si le graphe induit par l'ensemble non vide de sommets  $V'$  ne contient pas d'arêtes. On dira qu'un stable  $S$  de  $G$  est **maximal** lorsque le seul stable de  $G$  contenant  $S$  est  $S$  lui-même.

## 1 Un algorithme glouton – Langage C

### Allocation mémoire

Avec les notations précédentes, un graphe  $G$  sera représenté par la donnée de son nombre de sommets  $n$  et par sa matrice d'adjacence  $M$ . Ainsi  $M[i][j]$  vaudra 1 lorsque le graphe  $G$  contiendra l'arête  $\{i, j\}$  et 0 sinon.

On adoptera la structure suivante pour le représenter :

```
struct graph
{
    int n;
    int **M;
};
typedef struct graph graph;
```

**Question 1** Écrire une fonction de signature `graph graph_make(int n)` retournant un graphe à  $n$  sommets et sans arêtes.

**Question 2** Écrire une fonction de signature `void graph_free(graph g)` libérant la mémoire allouée à un graphe.

## Itérateur de permutations

On représente une permutation  $\sigma$  de  $\llbracket 0, n-1 \rrbracket$  par le tableau d'entiers  $[\sigma(0), \dots, \sigma(n-1)]$ .

On considère l'ordre des permutations de  $\llbracket 0, n-1 \rrbracket$  comme étant l'ordre lexicographique des tableaux qui les représentent :

$$[a_1] < [b_1] \Leftrightarrow a_1 < b_1$$
$$\forall p \geq 2, [a_1, \dots, a_p] < [b_1, \dots, b_p] \Leftrightarrow a_1 < b_1 \text{ ou } \begin{cases} a_1 = b_1 \\ [a_2, \dots, a_p] < [b_2, \dots, b_p] \end{cases}$$

Ainsi  $[2, 3, 1, 5, 4] < [2, 3, 4, 1, 5] < [2, 3, 4, 5, 1] < [2, 3, 5, 1, 4]$ .

**Question 3** Donner les 3 permutations qui suivent  $[2, 3, 5, 1, 4]$  dans l'ordre lexicographique.

**Question 4** Écrire une fonction de signature `int* identite(int n)` retournant la permutation identité de  $\llbracket 0, n-1 \rrbracket$ .

**Question 5** Écrire une fonction de signature `void inverse(int* p, int i, int j, int n)` modifiant la permutation de  $\llbracket 0, n-1 \rrbracket$  passée en paramètre. Cette fonction inversera l'ordre des éléments des cases dont l'indice appartient à  $\llbracket i, j \rrbracket$ . Cette fonction ne fera rien dans le cas où  $i \geq j$  et on supposera que cette fonction n'est appelée que lorsque  $i$  et  $j$  sont dans  $\llbracket 0, n-1 \rrbracket$ .

**Question 6** Écrire une fonction de signature `int indice_pps(int* p, int i, int n)` prenant en paramètre une permutation de  $\llbracket 0, n-1 \rrbracket$ , un entier  $i$  de  $\llbracket 0, n-1 \rrbracket$  et  $n$ . Cette fonction retournera, parmi les cases dont l'indice est strictement supérieur à  $i$  et dont la valeur est supérieure à  $p[i]$ , l'indice de la case de valeur minimale. On suppose que cette fonction ne sera appelée que lorsque cet indice existera.

**Question 7** Écrire une fonction de signature `bool suivante(int* p, int n)` qui modifie la permutation de l'ensemble  $\llbracket 0, n-1 \rrbracket$  passée en paramètre par la suivante dans l'ordre lexicographique. Lorsque la permutation suivante n'existe pas, la fonction renverra la valeur `true`; dans le cas contraire, elle renverra la valeur `false` sans modifier la permutation.

## Implémentation de l'algorithme glouton

On considère le graphe  $G_t = (V_t, E_t)$  représenté par la figure 1.

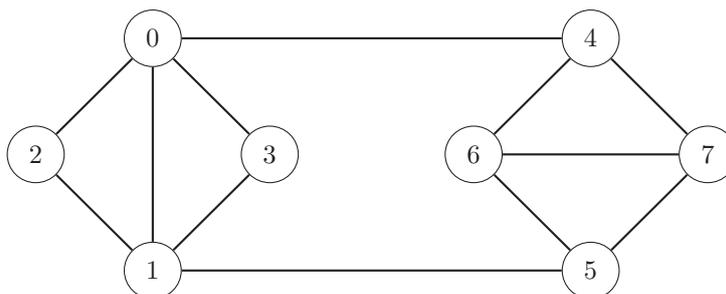


FIGURE 1 – Le graphe  $G_t$

On considère un algorithme glouton de coloration prenant en paramètre une permutation  $\sigma$  de  $\llbracket 0, n-1 \rrbracket$  et un graphe  $G$  à  $n$  sommets à colorer. L'algorithme parcourt le graphe dans l'ordre des sommets  $\sigma(0), \dots, \sigma(n-1)$  et colore chaque sommet non coloré par la couleur 0 si aucun de ses voisins n'est déjà coloré avec cette couleur. L'algorithme recommence alors avec la couleur suivante jusqu'à ce que tous les sommets soient colorés.

**Question 8** Appliquer l'algorithme glouton au graphe  $G_t$ .

**Question 9** Montrer que le graphe  $G_t$  est 3-colorable.

**Question 10** Écrire la fonction de signature `int* glouton(graph g, int* p)` prenant en paramètre un graphe non orienté  $g$  et une permutation de l'ensemble de ses sommets. Cette fonction retournera la coloration produite par l'algorithme glouton.

## Algorithme de coloration optimale

On fixe dans cette partie un graphe  $G = (V, E)$  à  $n \in \mathbb{N}^*$  sommets.

Pour chaque entier  $k \in \mathbb{N}^*$  et toute  $k$ -coloration, on définit  $(\star_k)$  la proposition suivante :  
 $\forall i \in \llbracket 0, k-1 \rrbracket$  « L'ensemble  $C_i$  des sommets de couleur  $i$  est un stable maximal du sous-graphe de  $G = (V, E)$  induit par les sommets  $V \setminus (C_0 \cup \dots \cup C_{i-1})$  »

**Question 11** En notant  $k$  le nombre de couleurs retourné par l'algorithme glouton sur une permutation des sommets de  $G$ , montrer que la coloration obtenue vérifie  $(\star_k)$ .

**Question 12** On considère une coloration à  $k$  couleurs satisfaisant  $(\star_k)$ . Montrer qu'elle est produite par l'exécution de l'algorithme glouton sur au moins une permutation des sommets de  $G$ .

**Question 13** Soit  $k \in \mathbb{N}^*$  le nombre minimum de couleurs des colorations de  $G$ . Montrer qu'il existe au moins une  $k$ -coloration vérifiant  $(\star_k)$ .

**Question 14** Écrire une fonction `int* coloration(graph g)` qui retourne une coloration optimale du graphe passé en paramètre.

## 2 Complexité

On considère le problème de décision **3-COLOR** suivant :

« Étant donné un graphe non orienté  $G$  à au moins 3 sommets, existe-t-il une 3-coloration de ses sommets ? »

On considère également le problème de décision **3-DOMINATION** suivant :

« Étant donné un graphe non orienté  $G$  à au moins 3 sommets, existe-t-il un ensemble de 3 sommets de  $G$  tel que chaque sommet de  $G$  soit à une distance au plus un de l'un d'eux ? »

Dans le cas où un tel ensemble existe, on dira qu'il s'agit d'un **ensemble 3-dominant** du graphe  $G$ .

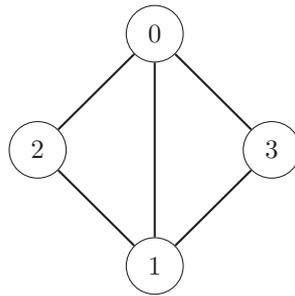
**Question 15** Montrer que 3-DOMINATION est dans  $P$ .

On considère à présent la construction suivante :

À partir d'un graphe  $G = (V, E)$ , on construit le graphe  $G' = (V', E')$  tel que :

1.  $V' = V \cup V_s$  où  $V_s$  est l'ensemble des stables de  $G$  ;
2.  $E'$  est l'union de l'ensemble de toutes les arêtes possibles entre les sommets de  $V_s$  et de l'ensemble des arêtes reliant les éléments de  $v \in V$  aux éléments de  $v_s \in V_s$  pour lesquels  $v$  est un sommet du stable  $v_s$ .

**Question 16** Représenter le graphe  $G'$  associé au graphe suivant :



**Question 17** Montrer qu'un graphe  $G$  admet une 3-coloration si et seulement si  $G'$  admet un ensemble dominant de taille 3.

**Question 18** Justifier que 3-COLOR est dans  $NP$ . Peut-on en déduire que  $P = NP$ ? On justifiera sa réponse.

### 3 Coloration distribuée – Langage OCaml

Dans cette partie on représente les graphes par liste d'adjacence au moyen du type `int list array`.

L'objectif à présent est de simuler l'exécution d'un algorithme de coloration de graphe distribué. On identifie dans ce cadre chaque sommet du graphe à une machine reliée à d'autres machines conformément aux arêtes du graphe.

Chaque sommet du graphe peut :

- exécuter des instructions ;
- connaître la liste de ses voisins ;
- envoyer des messages (informations) à un ou plusieurs de ses voisins ;
- lire les informations qu'il a reçues ;
- choisir sa propre couleur.

Les sommets ne peuvent donc pas accéder directement à l'information globale de coloration des autres sommets ; pour cela ils sont contraints de s'envoyer des messages.

Dans la mesure où nous effectuons une simulation dont seul le résultat nous intéresse, nous pourrions dans cette partie écrire des algorithmes sans ces limitations mais dont le résultat devra être le même que si ces contraintes étaient effectives. De plus, dans ce sujet lorsque l'on présentera un algorithme distribué, on considèrera que les sommets exécutent les instructions de façon synchrone : tous les sommets concernés par l'étape 1 exécutent les instructions de l'étape 1, puis tous les sommets concernés par l'étape 2 exécutent les instructions de l'étape 2, et ainsi de suite.

#### Un algorithme rapide de coloration

On considère à présent que les couleurs sont des entiers représentés par des mots binaires et on présente l'algorithme distribué `fast_color` suivant :

1. au départ aucun sommet n'est coloré et les sommets connaissent la liste de leurs voisins qu'ils considèrent comme actifs. Initialement, chaque sommet considère comme actifs chacun de ses voisins ;
2. chaque sommet non coloré génère aléatoirement et de façon uniforme un bit ;
3. chaque sommet non coloré envoie le bit généré à ses voisins encore considérés comme actifs ;
4. chaque sommet non coloré lit les bits reçus de ses voisins et ne considère plus comme actifs les voisins ayant généré un bit différent du sien ;
5. chaque sommet ne considérant plus aucun de ses voisins comme actif prend pour couleur l'entier représenté par le mot binaire obtenu en concaténant les bits qu'il a générés (du dernier bit généré au premier – de gauche à droite).
6. si il reste des sommets non colorés, on recommence à l'étape 2, sinon l'algorithme se termine.

On notera  $c(s)$  le nombre entier représentant la couleur produite par l'algorithme `fast_color` pour le sommet  $s$ .

**Question 19** Expliquer pourquoi, lorsque l'algorithme `fast_color` termine, deux sommets voisins ont nécessairement une couleur différente ?

**Question 20** Exécuter l'algorithme `fast_color` sur le graphe  $G_t$  (défini en figure 1) en considérant que l'on exécute les actions des sommets dans l'ordre croissant de leurs étiquettes et que les bits aléatoirement générés sont (de gauche à droite), les suivants : 001101110101101010. On pourra représenter par des pointillets les arêtes du graphe entre tout couple de sommets se considérant mutuellement inactifs.

**Question 21** Écrire une fonction de signature `bin_to_int : int list -> int` prenant un mot binaire représenté par une liste d'entiers de  $\{0, 1\}$  et retournant le nombre entier qu'il représente (on considèrera le bit de poids le plus fort comme étant le premier élément de la liste).

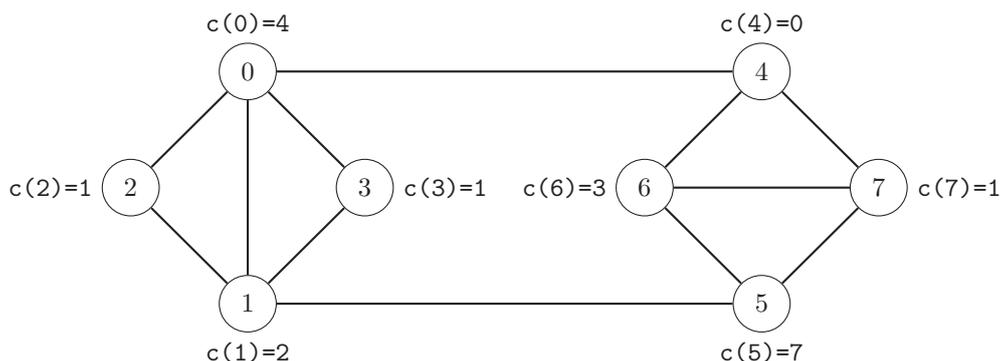
**Question 22** Écrire une fonction prenant en paramètre un graphe  $G$  représenté par la liste d'adjacence de ses sommets et une fonction générant un nombre entier aléatoire de  $\{0, 1\}$ . Cette fonction renverra un tableau indexé par les numéros des sommets du graphe  $G$  et dont les cases contiennent l'entier représentant sa couleur. La fonction aura pour signature `fast_color : int list array -> (unit -> int) -> int array` et on sera libre de créer des fonctions auxiliaires utiles.

## Un algorithme pour améliorer la coloration

Afin de réduire le nombre de couleurs utilisées par la coloration produite, on envisage un second algorithme distribué, appelé `reduce`, opérant après le précédent et visant à attribuer une nouvelle couleur aux sommets :

1. Chaque sommet stocke une liste  $\ell$ , initialement vide, des couleurs utilisées par ses voisins ;
2. Chaque sommet  $s$  sans nouvelle couleur envoie à ses voisins la valeur  $c(s)$  produite à l'issue de `fast_color` ;
3. Chaque sommet  $s$  sans nouvelle couleur et dont la valeur  $c(s)$  est inférieure à celle de chacun de ses voisins se voit attribuer comme nouvelle couleur la plus petite couleur absente de sa liste  $\ell$ . Il indique ensuite à tous ses voisins la couleur retenue ;
4. Chaque sommet sans nouvelle couleur ajoute à sa liste  $\ell$  la couleur de chacun de ses voisins s'étant vu attribué une nouvelle couleur à l'étape 3 ;
5. Tant qu'il reste des sommets sans nouvelle couleur, l'algorithme recommence à l'étape 2.

**Question 23** Exécuter l'algorithme `reduce` dans le cas où la coloration obtenue initialement était la suivante :



**Question 24** Écrire une fonction de signature `reduce : int list array -> int array -> int array` qui étant donné un graphe  $G$  représenté par sa liste d'adjacence et une coloration retournée par `fast_color` pour  $G$ , renvoie la coloration obtenue après exécution de l'algorithme distribué `reduce`. On sera libre de concevoir des fonctions auxiliaires aidant à l'élaboration de la fonction.

**Question 25** Montrer que l'algorithme distribué `reduce` termine.

**Question 26** Montrer que la coloration ainsi obtenue contient au plus  $\Delta + 1$  couleurs différentes où  $\Delta$  est le maximum des degrés des sommets du graphe.

Dans ce questionnaire à choix multiples, chaque question comporte une ou plusieurs bonnes réponses. Chaque réponse correcte fait gagner des points, mais chaque réponse fautive annule tous les points de la question. Les questions sont majoritairement formulées au pluriel par commodité d'expression. Cela n'implique pas nécessairement qu'elles admettent plusieurs réponses correctes.

1. Le nombre de langages sur l'alphabet  $\{a, b\}$  dont l'étoile est un langage fini est :
  - 0
  - 1
  - 2
  - infini
  
2. Parmi les langages suivants, indiquer ceux qui sont réguliers :
  - $\{a^n \mid n \in \mathbb{N}\}$
  - $\{a^n b^n \mid n \in \mathbb{N}\}$
  - $\{(ab)^n \mid n \in \mathbb{N}\}$
  - $\{a^n b^p \mid n \in \mathbb{N}, p \in \mathbb{N}, n + p = 2024\}$
  
3. Parmi les langages suivants, décrits de façon ensembliste ou dénotés à l'aide d'une expression régulière, sont engendrés par la grammaire :  $S \rightarrow bSS \mid S b S \mid S S b \mid a$  :
  - $\{\omega \in \{a, b\}^* \mid |\omega|_a > |\omega|_b\}$
  - $\{\omega \in \{a, b\}^* \mid |\omega|_a = |\omega|_b + 1\}$
  - $a(ba)^* \mid b(ab)^+$
  - $a(ba)^* \mid b(ab)^*$
  
4. Quelles grammaires, parmi les suivantes, permettent de produire le mot  $3p1t4$  sur l'alphabet  $\{d, 3, z, 1, p, 4, t\}$  :
  - $S \rightarrow 3S \mid S4 \mid pS \mid St \mid 1$
  - $S \rightarrow SSS \mid SS \mid 1 \mid 3 \mid 4 \mid p \mid t \mid z$
  - $S \rightarrow S1S \mid SpS \mid StS \mid 3 \mid 4$
  - $S \rightarrow SpS \mid StS \mid 1 \mid 3 \mid 4$
  
5. Quelles expressions régulières parmi les suivantes, dénotent un langage contenant le mot 314159 :
  - $((((1|9)(3|4)^*) ((1|3)(5|9)^*)^*)^*)^*$
  - $((((1|9)(3|4)^*)^* ((1|3)(5|9)^*)^*)^*)^*$
  - $((((1|9)(3|4)^*) ((1|3)(5|9)^*)^*)^*)^*$
  - $((((1|9)(3|4)^*)^* ((1|3)(5|9)^*)^*)^*)^*$
  
6. On considère un alphabet contenant au moins deux caractères. Indiquer parmi les propriétés suivantes celles qui sont vraies :
  - Il y a une infinité de langages réguliers.
  - Tout langage inclus dans un langage régulier est régulier.
  - Tout automate fini a pour langage reconnu un langage régulier.
  - Une intersection finie de langages réguliers est un langage régulier.

Fin du sujet