D'après Centrale 2015 - Option informatique

Les candidats devront répondre aux questions de programmation en utilisant le langage OCaml. Ils devront donner le type, ou la signature, de chaque fonction écrite, sauf lorsqu'il est indiqué par le sujet : dans ce cas la réponse doit être d'un type compatible avec la signature proposée.
Par exemple, la fonction cons définie par :

```
let cons x l = x::l ;;
```

est du type `'a -> 'a list -> 'a list` *qui est compatible avec la signature* `int -> int list -> int list`*.*
L'énoncé indique la signature attendue, toute réponse de type compatible est acceptée.

I Graphes d'intervalles

On considère le problème concret suivant : des cours doivent avoir lieu dans un intervalle de temps précis (de 8h à 9h55, et on cherche à attribuer une salle à chaque cours. On souhaite qu'à tout moment une salle ne puisse être attribuée à deux cours différents et on aimerait utiliser le plus petit nombre de salles possibles.

Ce problème d'allocation de ressources (ici les salles) en fonction de besoins fixes (ici les horaires des cours) intervient dans de nombreuses situations très diverses (allocation de pistes d'atterrissage aux avions, répartition de la charge de travail sur plusieurs machines, ...).

I.A – Représentation du problème

On modélise le problème ainsi :

- chaque besoin est représenté par un segment $[a, b]$ où $a, b \in \mathbb{N}$ et $a \leq b$.
- deux besoins I et J sont en conflit quand $I \cap J \neq \emptyset$.

La donnée du problème est une suite finie $(I_0, \ldots, I_{n-1})$ de n segments où $n \in \mathbb{N}$.

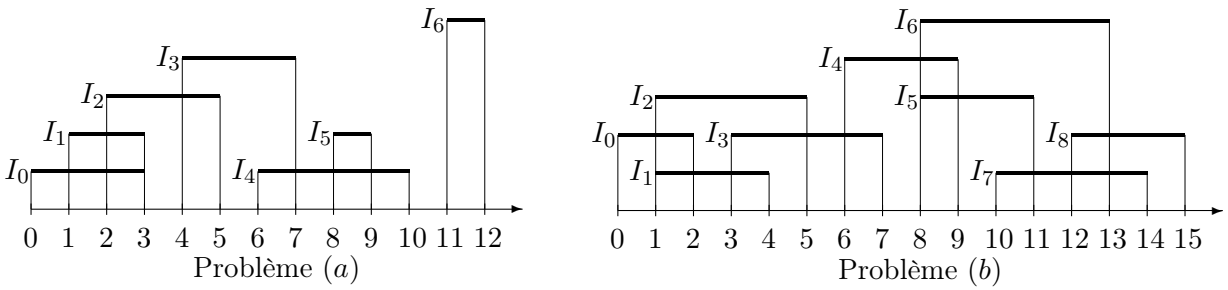

FIGURE 1 – Deux exemples de problèmes

On représente un segment en OCaml par un couple d'entiers, la donnée du problème est une valeur du type `(int*int) array`. Le problème (a) de la figure 1 est représenté par le tableau :

```
[| (0,3); (1,3); (2,5); (4,7); (6,10); (8,9); (11,12) |]
```

I.A.1 Écrire une fonction ayant pour signature `conflit : int * int -> int * int -> bool` telle que `conflit (a,b) (c,d)` renvoie `true` si et seulement si les intervalles $[a, b]$ et $[c, d]$ sont en conflit.

I.B – Graphe simple non orienté

On appelle graphe simple non orienté un couple $G = (S, A)$ où

- S est un ensemble fini dont les éléments sont appelés les sommets du graphe ;
- A est un ensemble de paires d'éléments distincts de S. Lorsque $\{x, y\} \in A$ on dit que x et y sont reliés dans G et $\{x, y\}$ est appelée une arête de G. Les sommets reliés à un sommet x sont appelés les voisins de x.

Étant donnée une énumération de S sous la forme d'une suite finie $(x_0, \dots, x_{n-1})$ on représente A en OCaml par un élément du type `int list array` ainsi : pour $i \in \{0, \dots, n-1\}$, la liste $A.(i)$ contient les j tels que x_i soit relié à x_j dans G.

On représente graphiquement le graphe G par un diagramme où les arêtes sont représentées par des traits entre les sommets.

Les arêtes du graphe dont une représentation graphique est donnée figure 2 sont représentées en OCaml par le tableau :

```
[| [1;2;3] ; [0;2;3] ; [0;1;3;4] ; [0;1;2] ; [2] |]
```

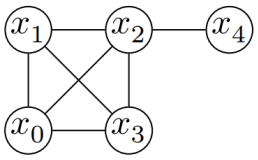

FIGURE 2 – Exemple de graphe

Un tel tableau de listes d'arêtes suffit pour déterminer un graphe lorsque l'énumération des sommets est connue car on peut alors identifier un sommet à son indice. Dans la suite de ce problème on identifiera ainsi un graphe à son tableau de listes d'arêtes.

I.C – Graphe d'intervalles

Soit $\overline{I} = (I_0, \dots, I_{n-1})$ une suite finie de segments, on appelle graphe d'intervalles associé à $\overline{I}$ le graphe $G(\overline{I})$:

- dont les sommets sont les segments $I_0, \dots, I_{n-1}$;
- et où, pour $i, j \in \{0, \dots, n-1\}$, avec $i \neq j$ les sommets I_i et I_j sont reliés si et seulement si ils sont en conflit.

Le graphe d'intervalles qui correspond au problème (a) de la figure 1 admet la représentation graphique de la figure 3.

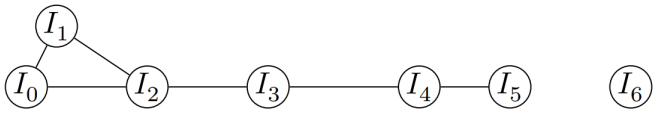

FIGURE 3 – Exemple de graphe d'intervalles

I.C.1 Donner une représentation graphique du graphe d'intervalles pour le problème (b) de la figure 1.

I.C.2 Écrire une fonction `construit_graphe : (int * int) array -> int list array` qui étant donné le tableau des segments $\overline{I} = (I_0, \dots, I_{n-1})$, énumérés dans cet ordre, renvoie la représentation des arêtes de G.

I.D – Coloration

Soit $G = (S, A)$ un graphe simple non orienté dont les sommets sont $x_0, \dots, x_{n-1}$. On appelle coloration de G une suite finie d'entiers naturels $(c_0, \dots, c_{n-1})$ telle que

$$\forall (i, j) \in \{0, \dots, n-1\}^2,\ \{x_i, x_j\} \in A \Rightarrow c_i \neq c_j$$

L'entier c_i est appelé la couleur du sommet x_i et la condition se traduit ainsi : deux sommets reliés ont des couleurs distinctes. Dorénavant, le terme couleur sera synonyme d'entier naturel.

La suite finie $(0, 1, 2, 3, 0)$ est une coloration du graphe de la figure 2.

Lorsqu'une coloration utilise le plus petit nombre de couleurs distinctes possibles, on dit qu'elle est optimale. On note alors $\chi(G)$ ce nombre minimum de couleurs, appelé le nombre chromatique de G.

En associant une salle à chaque couleur, on peut répondre au problème initial à l'aide d'une coloration de son graphe d'intervalles associé.

I.D.1 Déterminer des colorations optimales pour les graphes d'intervalles associés aux deux problèmes de la figure 1. On attribuera à chaque fois la couleur 0 à l'intervalle I_0.

I.D.2

a. Écrire une fonction `appartient : int list -> int -> bool` telle que `appartient l x` renvoie `true` si et seulement si l'entier x est présent dans la liste `l`.

b. Écrire une fonction `plus_petit_absent : int list -> int` telle que `plus_petit_absent l` renvoie le plus petit entier naturel non présent dans `l`.

c. On considère ici une coloration progressive des sommets d'un graphe. Pour cela, une coloration partielle est un tableau `couleurs : int array` tel que `couleurs.(i)` contient la couleur de i s'il est coloré et -1 sinon, ce qui ne pose pas de problème car les couleurs sont toujours positives.
Écrire une fonction `couleurs_voisins : int list array -> int array -> int -> int list` telle que `couleurs_voisins aretes couleurs i` renvoie la liste des couleurs des voisins colorés du sommet d'indice i dans le graphe décrit par `aretes` où le tableau couleurs décrit une coloration partielle.

d. En déduire une fonction `couleur_disponible : int list array -> int array -> int -> int` telle que `couleur_disponible aretes couleurs i` renvoie la plus petite couleur pouvant être attribuée au sommet i afin qu'il n'ait la couleur d'aucun de ses voisins dans le graphe décrit par `aretes`.

I.E – Cliques

Soit $G = (S, A)$ un graphe.
Un sous-ensemble $C \subset S$ est appelé une clique de G lorsqu'il vérifie :

$$\forall x, y \in C,\ x \neq y \Rightarrow \{x, y\} \in A$$

Le nombre d'éléments de C est appelé sa taille. La taille de la plus grande (celle qui possède le plus grand nombre d'éléments) clique de G est notée $\omega(G)$.

I.E.1 Déterminer $\chi(G)$ et $\omega(G)$ lorsque :

a. G ne possède pas d'arête (c'est à dire $A = \emptyset$).

b. G est un graphe complet à n sommets, c'est à dire $|S| = n$ et pour tous $u, v \in S$ distincts, $\{u, v\} \in A$.

I.E.2 Comparer $\chi(G)$ et $\omega(G)$ pour un graphe G quelconque.

I.E.3 Écrire une fonction `est_clique : int list array -> int list -> bool` telle que `est_clique aretes xs` renvoie `true` si et seulement si la liste `xs` est une liste d'indices de sommets formant une clique dans le graphe décrit par `aretes`.

II Graphes munis d'un ordre d'élimination parfait

On introduit ici la notion d'ordre d'élimination parfait, dont on montre qu'il existe toujours pour un graphe d'intervalles, et qui permet de proposer un algorithme glouton pour le problème de la coloration d'un graphe.

Soit $G = (S, A)$ un graphe et $(x_0, \dots, x_{n-1})$ une énumération des sommets de G.
Pour tout $i \in \{0, \dots, n-1\}$, on note $G_i = (S_i, A_i)$ où $S_i = \{x_0, \dots, x_i\}$ et :

$$\forall k, l \in \{0, \dots, n-1\},\ \{x_k, x_l\} \in A_i \iff (k, l \leq i \ \text{ et } \ \{x_k, x_l\} \in A)$$

G_i est ainsi le graphe déduit de G en se restreignant aux sommets de x_0 à x_i.

Une énumération $(x_0, \dots, x_{n-1})$ des sommets de G est appelée un ordre d'élimination parfait si pour tout $i \in \{0, \dots, n-1\}$, les voisins de x_i d'indices inférieurs à i forment une clique.

II.A – Un exemple

Déterminer un ordre d'élimination parfait pour le graphe G donné par la figure 4.

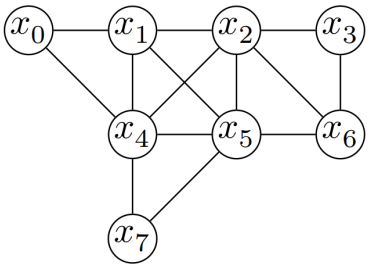

FIGURE 4 – Exemple de graphe pour ordre d'élimination parfait

II.B – Vérification

II.B.1 Écrire une fonction `voisins_inferieurs : int list array -> int -> int list` telle que `voisins_inferieurs aretes x` renvoie la liste des voisins du sommet d'indice `x` dont l'indice est strictement inférieur à `x`.

II.B.2 Écrire une fonction `est_ordre_parfait : int list array -> bool` telle que `est_ordre_parfait aretes` renvoie `true` si et seulement si l'énumération associée au graphe représenté par `aretes` est un ordre d'élimination parfait.

II.C – Coloration

Étant donnée une liste de segments $\overline{I} = (I_0, \dots, I_{n-1})$ de longueur $n \geq 1$, on se propose de déterminer une coloration optimale de son graphe d'intervalles associé. On appelle coloration de $\overline{I}$ une suite finie d'entiers naturels $(c_0, \dots, c_{n-1})$ telle que :

$$\forall (i,j) \in \{0, \dots, n-1\}^2,\ I_i \cap I_j \neq \emptyset \Rightarrow c_i \neq c_j$$

On considère un graphe dont $(x_0, \dots, x_{n-1})$ est une énumération des sommets.
On colore ce graphe à l'aide l'algorithme suivant :

pour i allant de 0 à $n-1$, on colore x avec la plus petite couleur qui ne soit pas utilisée par un de ses voisins déjà colorés.

II.C.1 Appliquer cet algorithme de coloration au graphe G de la figure 4 muni :

a. de l'ordre $(x_0, \dots, x_7)$;

b. d'un ordre d'élimination parfait.

III Algorithme glouton pour la coloration

On suppose dans cette partie que les segments $I_k = [a_k, b_k]$, pour $k \in \{0, \dots, n-1\}$, sont énumérés dans l'ordre croissant de leur extrémités gauches, c'est-à-dire que :

$$a_0 \leq a_1 \leq \cdots \leq a_{n-1}$$

On propose l'algorithme suivant :

Pour k variant de 0 à $n-1$, colorer l'intervalle I_k avec la plus petite couleur non encore utilisée dans la coloration des intervalles I_j, avec $0 \leq j < k$, qui ont une intersection non vide avec I_k.

Ainsi, l'intervalle I_0 est toujours coloré avec la couleur 0, l'intervalle I_1 reçoit la couleur 0 si $I_0 \cap I_1 = \emptyset$, et la couleur 1 sinon, etc.

III.A – L'algorithme sur un exemple

Déterminer la coloration renvoyée par l'algorithme pour le problème b décrit sur la figure 1.

III.B – Coloration

Écrire une fonction `coloration : (int * int) array -> int array` prenant en entrée un tableau contenant des segments triés par ordre croissant de leurs extrémités gauches, et renvoyant la coloration obtenue avec l'algorithme ci-dessus.

III.C – Preuve de l'algorithme

On se propose maintenant de démontrer que l'algorithme ci-dessus fournit une coloration optimale de l'ensemble de segments. Soit k un entier entre 0 et $n-1$. On suppose qu'à la k-ième étape de l'algorithme, le segment I_k reçoit la couleur c.

III.C.1 L'extrémité gauche du segment I_k appartient à un certain nombre de segments parmi $I_0, \dots, I_{k-1}$. Combien au moins ?

III.C.2 Prouver que l'ensemble constitué de I_k et de ses voisins d'indice inférieur à k constitue une clique de taille au moins $c+1$ dans le graphe d'intervalles associé.

Remarque. Cela prouve que l'ordre considéré dans cette partie est un ordre d'élimination parfait.

III.C.3 En déduire que le nombre de couleurs nécessaires à une coloration de l'ensemble des segments est au moins égal à $c+1$.

III.C.4 Conclure.

III.D – Complexité

Déterminer la complexité de la fonction `coloration` en fonction du nombre m d'arêtes du graphe d'intervalles associé à la liste $\overline{I}$, et du nombre n d'intervalles.

IV Ordre d'élimination parfait pour un graphe cordal

On s'intéresse ici à une nouvelle condition suffisante pour qu'un graphe admette un ordre d'élimination parfait, qui s'exprime en considérant les cycles de longueur au moins égale à 4 du graphe considéré.

Un graphe G est dit *cordal* lorsque pour tout cycle $C = (v_0, v_1, \dots, v_{n-1}, v_0)$ de G de longueur $n \geq 4$, il existe i, j distincts entre 0 et $n-1$ tels que les sommets v_i et v_j soient reliés dans le graphe G mais non successifs dans le cycle. Une telle arête $\{v_i, v_j\}$ est appelée une *corde* du cycle C. Autrement dit, le graphe G est cordal lorsque tout cycle de G de longueur supérieure ou égale à 4 possède une corde.

IV.A – Cycles de longueur 4 dans un graphe d'intervalles

Soit G un graphe d'intervalles. Dans cette question, on se propose de démontrer par l'absurde que tout cycle de longueur 4 de G possède une corde. On suppose à cet effet que G contient un 4-cycle sans corde.

On dispose donc de quatre segments I_0, I_1, I_2, I_3 tels que $I_0 \cap I_1 \neq \emptyset$, $I_1 \cap I_2 \neq \emptyset$, $I_2 \cap I_3 \neq \emptyset$, $I_3 \cap I_0 \neq \emptyset$. On supposera pour simplifier que les extrémités des segments sont toutes distinctes.

IV.A.1 Montrer qu'aucun des segments I_k, $k = 0, 1, 2, 3$, n'est inclus dans un autre de ces segments.

IV.A.2 On a donc par exemple $\min I_0 < \min I_1 < \max I_0 < \max I_1$.
Montrer que $\min I_1 < \min I_2 < \max I_1 < \max I_2$ et de même pour I_2 et I_3.

IV.A.3 Conclure à une contradiction.

IV.B – Cordalité des graphes d'intervalles

Montrer plus généralement que tout graphe d'intervalles est cordal.

IV.C – Une enquête policière

Six personnes sont entrées dans la bibliothèque le jour où un livre rare y a été volé. Chacune d'entre elles est entrée une seule fois dans la bibliothèque, y est restée un certain temps, puis elle en est sortie. Si deux personnes étaient ensemble dans la bibliothèque à un instant donné, alors au moins l'une des deux a vu l'autre. À l'issue de l'enquête, les témoignages recueillis sont les suivants : Albert dit qu'il a vu Bernard et Edouard dans la bibliothèque. Bernard a vu Albert et Fiona. Charlotte affirme avoir vu Didier et Fiona. Didier dit qu'il a vu Albert et Fiona. Edouard certifie avoir vu Bernard et Charlotte. Fiona dit avoir vu Charlotte et Edouard. Seul le coupable a menti. Qui est-il ?

IV.D – Ordre d'élimination parfait

Un sommet v d'un graphe G est dit simplicial lorsque l'ensemble des voisins de v dans G est une clique.

Étant donnés un graphe $G = (S, A)$ et $S' \subset S$ un ensemble de sommets de G, le *sous-graphe de G induit par S'* est le graphe $H = (S', A')$ où $A' \subset A$ est l'ensemble des arêtes de G dont les extrémités appartiennent à S'.

On représente en OCaml un sous-graphe induit d'un graphe G possédant n sommets par le couple `(aretes,sg)` de type `int list array * bool array` où `aretes` est une description du graphe G et `sg` est un tableau de taille n tel que `sg.(i)` vaut `true` si le sommet d'indice i est un sommet du sous-graphe induit et `false` sinon.

IV.D.1

a. Écrire une fonction `simplicial : (int list array * bool array) -> int -> bool` telle que `simplicial (aretes,sg) k`, où le sommet d'indice k est supposé appartenir au sous-graphe induit H décrit par `(aretes,sg)`, renvoie `true` si le sommet d'indice k est simplicial dans H et `false` sinon.

b. Déterminer la complexité de la fonction `simplicial`.

IV.D.2

a. Écrire une fonction `trouver_simplicial : (int list array * bool array) -> int` telle que `trouver_simplicial (aretes,sg)` renvoie, s'il en existe, un sommet simplicial du sous-graphe induit décrit par `(aretes,sg)`. Votre fonction lèvera l'exception `NoSimplicial` sinon.

b. Déterminer la complexité de cette fonction.

IV.D.3

a. Écrire une fonction `ordre_parfait : int list array -> int list` telle que `ordre_parfait aretes` renvoie un ordre d'élimination parfait du graphe décrit par `aretes`, s'il en existe un.

b. Déterminer la complexité de la fonction `ordre_parfait`.

IV.E – Coupures minimales dans un graphe cordal

Étant donné un graphe G on appelle *coupure* de G tout ensemble $C \subset S$ de sommets de G, de cardinal au moins égal à 2, tel que certains sommets reliés par un chemin dans le graphe G ne le sont plus dans le sous-graphe H de G induit par $S \setminus C$.

On se donne dans cette question un graphe cordal $G = (S, A)$. Soit C une coupure de G de cardinal minimal (supérieur ou égal à 2). Soit H le sous-graphe de G induit par $S \setminus C$. Soit a et b deux sommets de G déconnectés par la coupure, et soit G_1 et G_2 les composantes connexes de a et b dans le graphe H. Soit enfin x et y deux sommets distincts de la coupure C.

IV.E.1 Montrer que x est voisin dans le graphe G d'un sommet de G_1 et d'un sommet de G_2, et de même pour y.

IV.E.2 Montrer qu'il existe un chemin $P_1 = (x, a_1, \ldots, a_p, y)$ dont tous les sommets hormis x et y sont des sommets de G_1 et un chemin $P_2 = (x, b_1, \ldots, b_q, y)$ dont tous les sommets hormis x et y sont des sommets de G_2.

IV.E.3 On prend deux tels chemins P_1 et P_2 de longueur minimale. En considérant un cycle formé à partir des chemins P_1 et P_2, montrer que x et y sont voisins dans le graphe G.

IV.E.4 Montrer que C est une clique du graphe G.

IV.F – Sommets simpliciaux dans un graphe cordal

On se propose de montrer que tout graphe cordal G possède la propriété suivante, que l'on appellera la propriété $\mathcal{P}(G)$: *G possède un sommet simplicial, et même deux sommets simpliciaux non voisins si G n'est pas complet.* On se donne dans toute la question un graphe cordal G.

IV.F.1 Montrer que si G est complet alors tous ses sommets sont simpliciaux.

IV.F.2 Montrer que la propriété $\mathcal{P}(G)$ est vérifiée si G possède 1, 2 ou 3 sommets.

IV.F.3 On suppose dans cette question que G n'est pas complet, possède au moins trois sommets et que la propriété $\mathcal{P}(G)$ est vérifiée pour tous les graphes cordaux G' ayant strictement moins de sommets que G. Soit C une coupure de G de cardinal minimal. Soit a et b deux sommets de G déconnectés par la coupure C, et G_1 et G_2 les composantes connexes de a et b dans le sous-graphe de G induit par $S \setminus C$. Soit S_1 (resp. S_2) l'ensemble des sommets de G_1 (resp. G_2). Soit enfin H_1 (resp. H_2) le sous-graphe de G induit par $S_1 \cup C$ (resp. $S_2 \cup C$).

a. Justifier que le graphe H_1 est cordal.

b. On suppose que H_1 est complet. Montrer que S_1 contient un sommet simplicial du graphe H_1. Prouver que ce sommet est en fait un sommet simplicial de G.

c. On suppose que H_1 n'est pas complet. Montrer que $S_1 \cup C$ contient deux sommets simpliciaux non voisins du graphe H_1. Montrer que au moins l'un de ces deux sommets est dans S_1 et que ce sommet est un sommet simplicial de G.

d. Montrer que la propriété $\mathcal{P}(G)$ est vérifiée.

IV.G – Ordre d'élimination parfait pour un graphe cordal

Montrer que tout graphe cordal possède un ordre d'élimination parfait.