DM2 : Graphes d'intervalles et coloration

1. **Graphes d'intervalles.**

   On considère le problème concret suivant : des cours doivent avoir lieu dans un intervalle de temps précis (de 8h00 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, ...).

   (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 $n \in \mathbb{N}$.

   Voici deux exemples de problèmes :

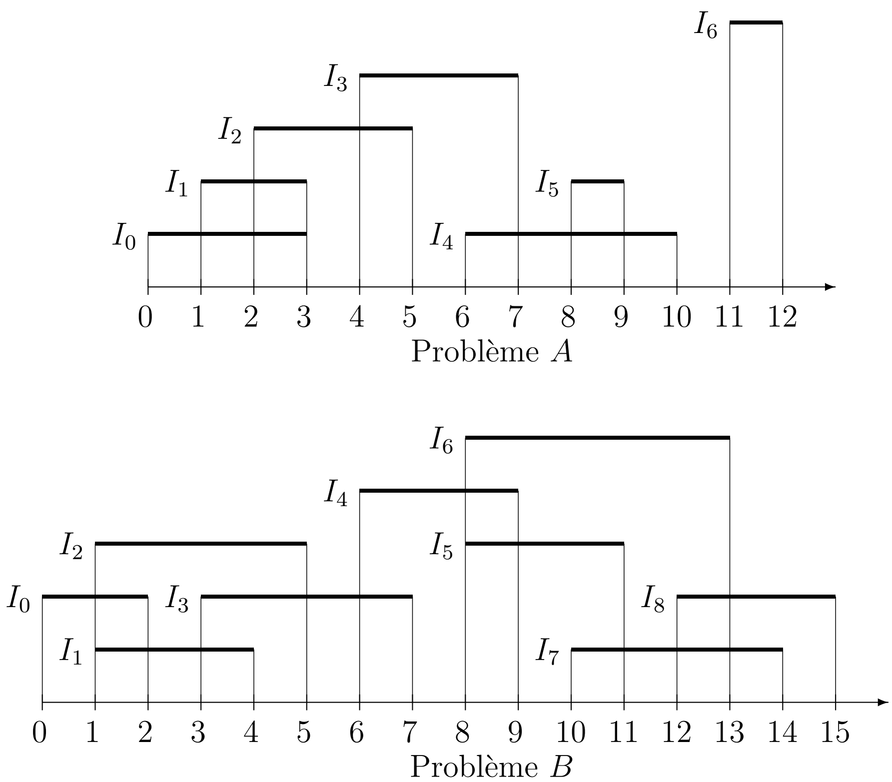

FIGURE 1 : Deux exemples de problèmes.

   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) |]
```

Écrire une fonction ayant pour signature

$$\texttt{conflit : int * int} \to \texttt{int * int} \to \texttt{bool}$$

telle que `conflit I J` renvoie `true` si et seulement si I et J sont en conflit.

(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 $(x_0, \ldots, x_{n-1})$, on représente A en OCaml par un élément du type `int list array`. Ainsi, pour $i \in \{0, \ldots, n-1\}$, 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.

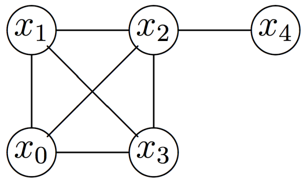

FIGURE 2

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

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

Les listes des voisins ne sont pas nécessairement triées par ordre croissant.

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.

(c) **Graphe d'intervalles.**

Soit $I = (I_0, \ldots, I_{n-1})$ une suite finie de segments. On appelle graphe d'intervalles associé à I le graphe $G(I)$ tel que :

- les sommets sont les segments $I_0, \ldots, I_{n-1}$,
- pour $i, j \in \{0, \ldots, n-1\}$, 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 :

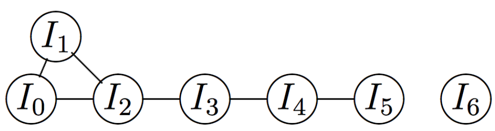

FIGURE 3

i. Donner une représentation graphique du graphe d'intervalles pour le problème B de la figure 1.

ii. Écrire une fonction itérative ayant pour signature

`construit_graphe : (int * int) array → int list array`

qui étant donné le tableau des segments $I = (I_0, \ldots, I_{n-1})$, énumérés dans cet ordre, renvoie la représentation des arêtes de G.

Notons que pour une variable `A` de type `int list array`, les éléments sont des listes et sont mutables, ce qui nous permet : `A.(i) <- t::A.(i)` .

(d) **Coloration.**

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

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

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. 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.

ii. Écrire une fonction récursive de signature

`appartient : int list → int → bool`

telle que l'appel `appartient l x` renvoie `true` si et seulement si l'entier x est présent dans la liste `l`.

iii. Écrire une fonction itérative de signature

`plus_petit_absent : int list → int`

telle que l'appel `plus_petit_absent l` renvoie le plus petit entier naturel non présent dans `l`.

iv. 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 de signature

`couleurs_voisins : int list array → int array → int → int list`

telle que l'appel `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.

La fonction `couleurs_voisins` pourra inclure une fonction auxiliaire récursive terminale et se limiter à l'appel de cette fonction auxiliaire.

v. En déduire, une fonction de signature

`couleur_disponible : int list array` $\rightarrow$ `int array` $\rightarrow$ `int` $\rightarrow$ `int`

telle que l'appel `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`.

(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. Déterminer $\chi(G)$ et $\omega(G)$ lorsque
   - G ne possède pas d'arête (c'est-à-dire $A = \emptyset$).
   - G est un graphe complet à n sommets, c'est-à-dire $|S| = n$ et pour tous $u, v \in S$ distincts, $\{u, v\} \in A$.

ii. Comparer $\chi(G)$ et $\omega(G)$ pour un graphe G quelconque.

iii. Écrire une fonction récursive de signature

`est_clique : int list array` $\rightarrow$ `int list` $\rightarrow$ `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`.

2. **Algorithme glouton pour la coloration.**

Étant donnée une liste de segments $I = (I_0, \dots, I_{n-1})$ de longueur $n \geq 1$, on se propose de déterminer une coloration optimale de son graphe d'intervalles associé.

On appelle coloration de 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 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 leurs extrémités gauches, c'est-à-dire que

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

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 ...

(a) **L'algorithme sur un exemple.**

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

(b) **Coloration.**

Écrire une fonction itérative de signature

`coloration : (int * int) array → int list array → int array`

telle que l'appel `coloration segments aretes`, où `segments` est un tableau contenant des segments triés par ordre croissant de leurs extrémités gauches et où `aretes` représente les arêtes du graphe d'intervalles associé à ces segments, renvoie la coloration obtenue avec l'algorithme ci-dessus.

(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.

i. L'extrémité gauche du segment I_k appartient à un certain nombre de segments parmi $I_0, \ldots, I_{k-1}$. Combien au moins ?
ii. 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.
iii. En déduire que le nombre de couleurs nécessaires à une coloration de l'ensemble des segments est au moins égal à $c+1$.
iv. Conclure.

(d) **Complexité.**

Déterminer la complexité de la fonction `coloration` en fonction du nombre m d'arêtes du graphe d'intervalles associé à la liste I.

Cette détermination vous amènera à évaluer la complexité de toutes les fonctions utilisées comme `appartient`, `plus_petit_absent`, `couleurs_voisins`, `couleur_disponible`.