DS 1

Option informatique, deuxième année

Julien Reichert

Durée : 4 heures.

Le langage Caml est imposé pour l'exercice 1, et remplaçable par du pseudo-code bien lisible pour l'exercice 2. On impose de donner le type, ou la signature, de chaque fonction écrite, sauf lorsqu'il ou elle est indiqué(e) par le sujet, auquel 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.

1 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,...).

1.1 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 $n \in \mathbb{N} \setminus \{0\}$.

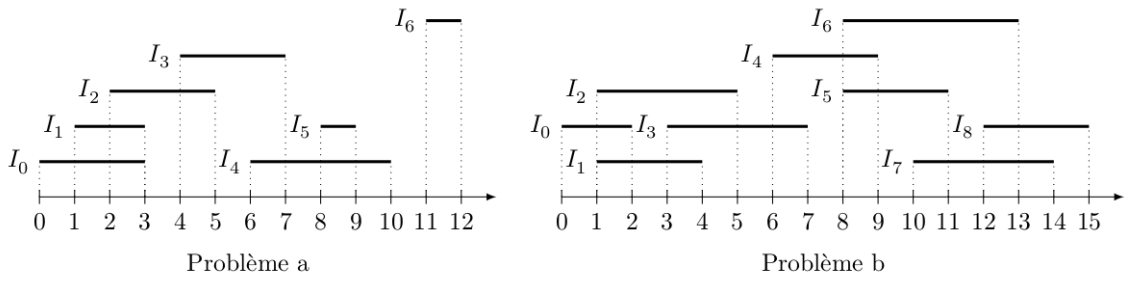

Figure 1 Deux exemples de problèmes

On représente un segment en Caml par un couple d'entiers, la donnée du problème est une valeur du type `(int*int) vect`. 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) |].
```

Question 1 : Écrire une fonction ayant pour signature `conflit : int * int -> int * int -> bool` telle que `conflit I J` renvoie `true` si et seulement si `I` et `J` sont en conflit.

1.2 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 Caml par un élément du type `int list vect` 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.

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

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

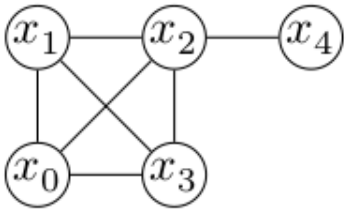

Figure 2

Une telle liste 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 sa liste d'arêtes.

1.3 Graphe d'intervalles

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

— dont les sommets sont les segments I_0 , …, I_{n-1} ;
— et où, 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 de la figure 3.

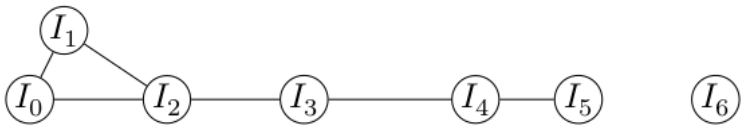

Figure 3

Question 2 : Donner une représentation graphique du graphe d'intervalles associé au problème b de la figure 1.

Question 3 : Écrire une fonction ayant pour signature `construit_graphe : (int * int) vect -> int list vect` qui étant donné le tableau des segments $\bar{I} = (I_0, \ldots, I_{n-1})$, énumérés dans cet ordre, renvoie la représentation des arêtes de $G(\bar{I})$.

1.4 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

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

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.

Question 4 : 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.

Question 5 : Écrire deux fonctions, l'une de signature `appartient : int list -> int -> bool`, telle que l'appel à `appartient l x` envoie `true` si et seulement si l'entier `x` est présent dans la liste `l`, et l'autre de signature `plus_petit_absent : int list -> int`, telle que l'appel à `plus_petit_absent l` renvoie le plus petit entier naturel non présent dans `l`.

Question 6 : On considère ici une coloration progressive des sommets d'un graphe. Pour cela, une coloration partielle est un tableau `couleurs : int vect` 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 vect -> int vect -> int -> int list` telle que l'appel à `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.

Question 7 : En déduire une fonction de signature

```
couleur_disponible : int list vect -> int vect -> int -> int
```

telle que l'appel à `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`.

1.5 Cliques

Soit $G = (S, A)$ un graphe. Un sous-ensemble $C \subseteq S$ 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 $\omega(G)$.

Question 8 : Déterminer $\chi(G)$ et $\omega(G)$ lorsque G ne possède pas d'arête (c'est à dire $A = \emptyset$), et lorsque G est un graphe complet à n sommets, c'est à dire $|S| = n$ et pour tous $u, v \in S$ distincts, $\{u, v\} \in A$.

Question 9 : Comparer $\chi(G)$ et $\omega(G)$ pour un graphe G quelconque.

Question 10 : Écrire une fonction de signature `est_clique : int list vect -> 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`.

1.6 Algorithme glouton pour la coloration

Étant donnée une liste de segments $\bar{I} = (I_0, I_1, \ldots, I_{n-1})$ de longueur $n \geq 1$, on se propose de déterminer une coloration optimale de son graphe d'intervalles associé. On appelle coloration de $\bar{I}$ une suite finie d'entiers naturels $(c_0, \ldots, c_{n-1})$ telle que

$$\forall i, j \in \{0, \ldots, n-1\}, I_i \cap I_j \neq \emptyset \Rightarrow c_i \neq c_j.$$

On suppose dans cette partie que les segments $I_k = [a_k, b_k]$, pour $k \in \{0, \ldots, n-1\}$, 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.

Question 11 : Déterminer la coloration renvoyée par l'algorithme pour le problème b décrit sur la figure 1. Si par hasard elle a déjà été donnée à la question 4, on peut se contenter de le signaler tout simplement.

Question 12 : Écrire une fonction de signature `coloration : (int * int) vect -> int list vect -> int vect` 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.

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.

Question 13 : L'extrémité gauche du segment I_k appartient à un certain nombre de segments parmi $I_0, I_1, \ldots, I_{k-1}$. Combien au moins ?

Question 14 : 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.

Question 15 : En déduire que le nombre de couleurs nécessaires à une coloration de l'ensemble des segments est au moins égal à $c+1$, puis conclure.

Question 16 : Déterminer la complexité de la fonction `coloration` en fonction du nombre m d'arêtes du graphe d'intervalles associé à la liste $\bar{I}$.

2 Calculs de temps de trajet

Question 1 : Décrire une structure de données S, pour stocker des éléments pondérés, qui permet les opérations suivantes :

- `insere(S,elem,w)` ajoute un élément `elem` de poids `w` à la structure `S` ;
- `recherche_min(S)` retourne un élément de poids minimum dans `S` ainsi que son poids ;
- `efface_min(S)` enlève l'élément de poids minimum de `S`.

Les temps de calcul de `insere` et `efface_min` doivent être en $O(\log n)$ où n est le nombre d'éléments dans la structure au moment de l'appel ; `recherche_min` doit fonctionner en $O(1)$. On peut supposer que les poids sont entiers. Donner un argument bref expliquant que les fonctions se déroulent dans les temps souhaités et produisent des résultats corrects.

Question 2 : Combien de temps faut-il pour aller de la ville a à la ville z si le temps de voyage entre les villes est décrit ci-après ?

- 1 minute entre les villes a et b ;
- 9 minutes entre les villes a et c ;
- 4 minutes entre les villes a et d ;
- 5 minutes entre les villes b et e ;
- 12 minutes entre les villes b et f ;
- 4 minutes entre les villes c et f ;
- 4 minutes entre les villes c et g ;
- 2 minutes entre les villes d et g ;
- 8 minutes entre les villes e et h ;
- 3 minutes entre les villes f et h ;
- 7 minutes entre les villes f et z ;
- 2 minutes entre les villes f et i ;
- 3 minutes entre les villes h et z ;
- 6 minutes entre les villes i et z.

Question 3 : En utilisant la structure de données décrite à la question 1, écrire un algorithme qui prend en entrée une liste de triplets d'entiers `(d,t,a)` décrivant le temps de voyage `t` entre deux villes `d` et `a` et qui calcule le temps minimum qu'il faut pour aller de la ville `s` à la ville `f`, toutes deux en paramètre. Chaque ville est représentée par un numéro entre 1 et 100000.

Pour la question précédente, si le format de l'entrée de l'algorithme ne convient pas, il est permis d'en utiliser un autre à condition de le préciser, de le justifier et d'écrire une fonction permettant de passer du format de l'énoncé au format choisi (si le temps le permet).

Question 4 : Maintenant, supposons que traverser une ville (c'est-à-dire arriver et repartir) prend aussi du temps. Le temps requis est la moitié, arrondie par défaut à la minute, du nombre de minutes restantes jusqu'à la prochaine heure. Donc, si on arrive à 2 h 16 on sortez de la ville $\frac{44}{2} = 22$ minutes plus tard. Arriver pile à l'heure permet de sortir immédiatement. Décrire un algorithme pour trouver le temps minimum requis pour arriver à la ville z de la ville a en quittant celle-ci à minuit. Donner la réponse pour cet exemple.