Programmation fonctionnelle avec OCaml
TP 6

Objectifs :
— modularité ;
— abstraction de type ;
— utilisation de foncteurs.

Cliques maximales d'un graphe d'intervalles

Un *graphe d'intersection* $G = (V, E)$ est un graphe défini par un ensemble d'objets X et une relation d'intersection entre les objets inter : $X^2 \to \mathbb{B}$ tel que $V = \{v_i, \forall x_i \in X\}$ et $E = \{(v_i, v_j), \forall x_i, x_j \in X^2, i \neq j,$ t.q. inter$(x_i, x_j)\}$, i.e. chaque nœud correspond à un objet et une arête relie deux nœuds si les objets correspondants s'intersectent. Un *graphe d'intervalles* est le graphe d'intersection d'un ensemble d'intervalles de la droite réelle, i.e. $\forall x_i \in X$, $x_i = [s_i, e_i[$ avec $s_i, e_i \in \mathbb{R}$, $s_i < e_i$:

$$V = \{v_i, \forall x_i \in X\}, \quad E = \{(v_i, v_j), \forall x_i, x_j \in X^2, i \neq j, x_i \cap x_j \neq \varnothing\}$$

Si un intervalle correspond à l'exécution d'une tâche sur une ressource de capacité unitaire (e.g. l'occupation d'une porte d'aéroport par un avion), l'allocation des vols aux portes correspond à la coloration du graphe (en associant une couleur à chaque porte). On peut obtenir le minimum de portes nécessaire à l'allocation en calculant la *clique maximum*, i.e. le plus grand sous-ensemble de nœuds tous reliés entre eux deux à deux, donc le plus grand $V' \subseteq V$ tel que $\forall v_i, v_j \in V'^2$, $i \neq j$, $(v_i, v_j) \in E$.

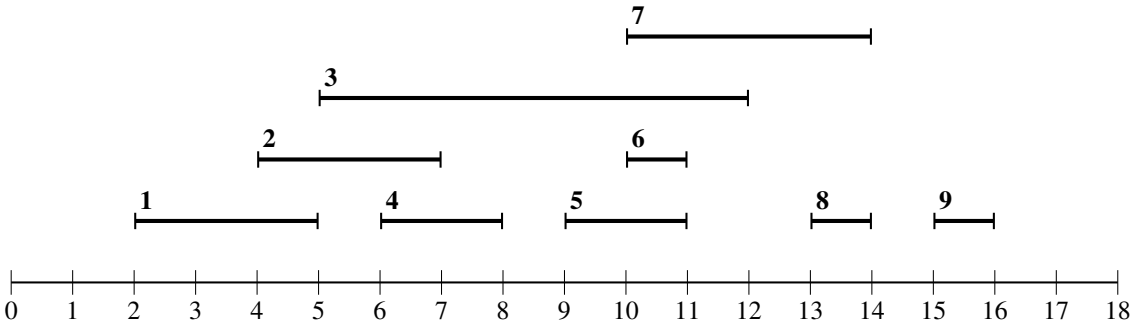

FIGURE 1 – Une instance à 9 tâches avec 5 cliques maximales (et une clique maximum de taille 4) : $\{1, 2\}$, $\{2, 3, 4\}$, $\{3, 5, 6, 7\}$, $\{7, 8\}$, $\{9\}$.

Une *clique maximale* est une clique qui n'est contenue strictement dans aucune autre clique ; la *clique maximum* est donc la plus grande *clique maximale*. On va écrire un algorithme qui calcule *toutes les cliques maximales* d'un graphe d'intervalles à partir d'une liste de n intervalles (ou tâches). L'algorithme balaie les extrémités des tâches de gauche à droite, et construit une clique (initialement vide) en ajoutant une tâche à la clique à chaque début d'intervalle et la retirant à chaque fin. Quand le cardinal de la clique passe par un maximum (i.e. une tâche doit être retirée après une séquence d'ajouts), la clique est maximale et on l'ajoute à la liste des cliques à renvoyer. Plus précisément :

1. On commence par créer la liste des $2n$ *extrémités* (s_i et e_i) des intervalles, puis on la trie par date croissante.
2. On parcours cette liste récursivement en initialisant la clique maximale en construction à l'ensemble vide, puis pour chaque extrémité :
   (a) S'il s'agit d'un début de tâche, on ajoute la tâche correspondante à la clique maximale en construction et on effectue l'appel récursif.
   (b) S'il s'agit d'une fin de tâche :

i. Si c'est la première fin de tâche rencontrée après une séquence d'ajouts de tâches (une seule ou plusieurs) à la clique en construction, on a détecté une nouvelle clique maximale et on doit l'ajouter à la liste résultat.

ii. Dans tous les cas, il faut supprimer la tâche correspondante de la clique en construction, puis effectuer l'appel récursif.

(c) On s'arrête quand la liste des extrémités est épuisée.

Sur l'instance de la Figure 1, cet algorithme produit 5 cliques maximales. Dans la suite, on considérera (sans perte de généralité) que les dates sont entières, i.e. $\forall x_i \in X,\ s_i, e_i \in \mathbb{N}^2$.

1. Dans le fichier `task.ml` fourni, définir un type enregistrement `task` qui contient trois champs entiers : l'identité de la tâche, son début et sa fin. Définir également une fonction d'impression `fprint_task: out_channel -> task -> unit` et une fonction `make: int -> int -> int -> task` telle que `make id start endt` renvoie la tâche d'identité `id`, début `start` et fin `endt`.
2. Définir un module `TS` d'ensemble de tâches à l'aide du foncteur du module `Set`. Les tâches seront comparées uniquement selon leur identité. Renommer ensuite le type `TS.t` en type `t` et la fonction `TS.fold` en fonction `fold` (pour pouvoir les exporter ultérieurement dans le fichier d'interface). La fonction `read: string -> t` fournie renvoie l'ensemble de tâches lues dans un fichier de données et la fonction `fprint: out_channel -> t -> unit` imprime un ensemble de tâches.
3. Créer le fichier d'interface `task.mli` où l'on exportera :
   — le type `task` (sans abstraction) ;
   — let type `t` abstrait ;
   — les fonctions `read`, `fprint` et `fold` en utilisant les deux types précédents.
4. Dans un nouveau fichier `event.ml`, définir un type enregistrement `t` pour représenter les extrémités des tâches avec trois champs :
   — l'identité (un entier) de la tâche correspondante ;
   — la date (un entier) ;
   — le type d'extrémité (début ou fin de tâche), en utilisant un type somme `kind` avec deux constructeurs constants (i.e. sans paramètre) à définir au préalable.
   Définir également une fonction d'impression pour ce type `fprint: out_channel -> t -> unit`.
5. Définir la fonction `is_start: t -> bool` qui indique si son paramètre correspond à un début ou une fin de tâche, et la fonction `id: t -> int` qui renvoie l'identité de son paramètre.
6. En cas d'égalité de date, il faut comparer les champs de type `kind` des extrémités pour tenir compte du fait que les intervalles sont fermés à gauche et ouverts à droite, donc **une fin de tâche est inférieure à un début de tâche**. Écrire une fonction `compare_kind: kind -> kind -> int` qui renvoie -1 si le premier paramètre est inférieur au second, 0 s'ils sont égaux et 1 sinon.
7. Écrire une fonction `compare: t -> t -> int` équivalente à la fonction `compare_kind` mais pour les extrémités : on utilisera la fonction `compare` générique de la bibliothèque standard pour comparer les dates et, en cas d'égalité, la fonction `compare_kind`. Veillez à n'effectuer aucun calcul inutile.
8. Écrire une fonction `tasks2events: Task.t -> t list` qui prend un ensemble de (n) tâches en paramètre et renvoie la liste des ($2n$) extrémités correspondantes triées (dans l'ordre croissant) selon la fonction `compare`. On utilisera `Task.fold` pour construire la liste des extrémités avant de la trier.
9. Créer le fichier d'interface `event.mli` où seront exportés :
   — le type `t` abstrait ;
   — les fonctions `fprint`, `id`, `is_start` et `tasks2events`.
10. Dans le fichier `clique.ml` fourni, définir le module `IS` des ensembles d'entiers à l'aide du foncteur du module `Set`. Une clique sera représentée avec le type `IS.t`, i.e. avec l'ensemble des identités des tâches qui la constitue. Renommer le type `IS.t` en type `t` et définir la fonction `size: t -> int` qui renvoie le nombre d'éléments d'une clique (en temps constant). La fonction `fprint: out_channel -> t -> unit` permet d'imprimer une clique.

11. Écrire la fonction `maximals: Task.t -> t list` qui commence par transformer un ensemble de tâches en liste triée d'extrémités puis les parcourt avec une *fonction récursive locale* pour renvoyer la liste de toutes les cliques maximales :
    — En plus des extrémités, la fonction locale prendra en paramètre la clique en construction et un booléen qui indique si le point précédent correspondait à un ajout (début de tâche) ou un retrait (fin de tâche).
    — Si la liste n'est pas vide :
        — S'il s'agit d'un début de tâche, on effectue l'appel récursif en ajoutant l'identité de la tâche à la clique et avec le booléen à vrai.
        — Si c'est une fin de tâche, et que le booléen est vrai, on est passé par un maximum local et il faut ajouter la clique à la liste résultat. Que le booléen soit vrai ou faux, il faut réaliser l'appel récursif en supprimant la tâche de la clique en construction et avec le booléen à faux.
12. Écrire la fonction `maximum: t list -> t` qui prend en paramètre la liste de toutes les cliques maximales et renvoie la clique maximum, en utilisant l'itérateur approprié du module `List`.
13. Créer le fichier d'interface `clique.mli` en exportant :
    — le type `t` abstrait ;
    — les fonctions `fprint`, `size`, `maximals` et `maximum` (en utilisant le type précédent).
14. Créer le module principal `main.ml` et écrire un (pseudo-)main pour calculer et afficher toutes les cliques maximales de l'exemple contenu dans le fichier `tasks.txt` fourni (correspondant à la Figure 1) :

```
barnier@knuth:exam>./maxclique tasks.txt
cliques maximales:
{1,2}
{2,3,4}
{3,5,6,7}
{7,8}
{9}
clique maximum: {3,5,6,7} cardinal=4
```