TD n°16 – Algorithmes gloutons

Exercice 1 *Ordonnancement d'intervalles*

Dans ce problème, on considère un ensemble d'intervalles $S = \{]s_i, f_i[, 0 \leq i \leq n-1\}$. On veut choisir le plus grand sous-ensemble $S' \subset S$ tel que les intervalles de S' soient deux à deux disjoints.

Par exemple pour $S = \{]0,5[,]2,7[,]6,10[\}$, on voit que le deuxième intervalle intersecte le premier et le troisième. La solution optimale est donc $S' = \{]0,5[,]6,10[\}$, de taille 2. $S' = \{]2,7[\}$ est aussi une solution possible, mais elle n'est **pas optimale** car elle est de taille 1.

On considère l'approche gloutonne suivante :

- On part de $S' = \emptyset$.
- Tant que c'est possible, on regarde l'ensemble P des intervalles de S qui ne sont pas dans S' et qui sont disjoints des intervalles de S'.
  On choisit ensuite un intervalle I dans P, en suivant une certaine règle et on l'ajoute à S' .

Plusieurs règles (ou heuristiques) sont envisageables :

- Choisir l'intervalle de P qui commence le plus tôt.
- Choisir l'intervalle de P le plus court (i.e. tel que $f_i - s_i$ est minimal).
- Choisir l'intervalle de P avec le f_i le plus petit.

1. Montrer que les deux premières règles peuvent ne pas fournir une solution optimale. (Il suffit de trouver un exemple où le S' obtenu n'est pas le plus grand possible).

On veut montrer que la troisième heuristique est optimale.

On note $G = (i_1, ..., i_m)$ avec $\forall j \in [|1,m|], 0 \leq i_j \leq n-1$ la famille d'intervalles choisis par un algorithme glouton selon la troisième règle. On les considère dans l'ordre de choix, c'est à dire ordonnés par horaire de fin croissant.

On considère une autre famille quelconque $H = (j_1, ..., j_p)$ d'intervalles de S deux à deux disjoints, avec $p \geq m$. Cela signifie que H est une solution qui est aussi bien ou meilleure que G.

2. Montrer par récurrence sur $l \in [|1,m|]$ que pour tout $1 \leq r \leq l$, $f_{i_r} \leq f_{j_r}$.
3. En déduire que $m = p$, c'est à dire que H ne peut pas être meilleure que G. Conclure sur la 3ème règle.

On va maintenant implémenter la méthode en Ocaml

4. Proposer un type Ocaml pour stocker chacun des intervalles i.
5. Écrire une fonction `disjoints` qui détermine si deux intervalles sont disjoints

On pourra utiliser la fonction `List.sort : ('a->'a->int)-> a list -> 'a list` qui prend en entrée une fonction de comparaison et une liste et renvoie la liste triée dans l'ordre croissant défini par la fonction de comparaison.

La fonction de comparaison prend deux éléments et renvoie 0 s'ils sont égaux, un nombre positif si le premier est plus grand et un nombre négatif sinon.

Par exemple, pour trier selon les couples, on peut écrire la fonction de comparaison suivante :

```
let compare_couples (a,b) (c,d) =
 if a = c && b = d then 0
 else if a > c || (a=c && b>d) then 1
 else -1;;
```

6. Écrire une fonction qui trie une liste d'intervalles selon la troisième règle.
7. Écrire l'algorithme glouton complet.
8. En considérant que `List.sort` s'effectue en temps optimal, quelle est la complexité de cette méthode ?

Exercice 2 *Partitionnement d'intervalles*

Dans ce problème on considère la même entrée que dans le précédent. Le but est cette fois de trouver le k le plus petit tel qu'on peut partitionner les intervalles de S en k parties telles que deux intervalles d'un même partie sont toujours disjoints.

Par exemple pour $S = \{]0,5[,]2,7[,]6,10[\}$, on peut partitionner en $\{]0,5[,]6,10[\}$ et $\{]2,7[\}$, avec deux classes, ce qui est optimal. On peut également partitionner en $\{]0,5[\}$, $\{]6,10[\}$ et $\{]2,7[\}$, avec trois classes, ce qui est **sous-optimal**.

Ce problème modélise par exemple la réalisation d'emploi du temps : on cherche de combien de salles k on a besoin au minimum pour pouvoir organiser une liste de cours.

On considère l'algorithme glouton suivant :

- Ranger les n intervalles dans un certain ordre $[r_1, ..., r_n]$. Cela peut être fait selon une règle précise (par date de début croissante, ...), au hasard ou en conservant juste l'ordre que les intervalles ont initialement.
- Attribuer l'intervalle 1 à la partie 1.
- Initialiser un $d = 1$, pour se rappeler du nombre de parties utilisées jusqu'à maintenant.
- Pour i de allant de 2 à n :
  - Chercher le plus petit entier $k \leq d$ tel que l'on puisses attribuer l'intervalle r_i à la partie k.
  - Si il en existe un : mettre l'intervalle r_i dans la partie k.
  - Sinon : mettre l'intervalle r_i dans une nouvelle partie de numéro $d + 1$, et incrémenter d.
- Renvoyer la partition construite ou d selon ce qu'on veut avoir.

1. Montrer que peu importe l'ordre choisi pour la première étape, cette méthode renvoie une partition telle que les intervalles d'une même partie sont disjoints.
2. Exhiber un exemple sur lequel l'algorithme ne renvoie pas la solution optimale. On pourra considérer que l'ordre de rangement à l'étape 1 est de conserver l'ordre d'entrée (donc ne rien faire).

On va maintenant montrer que si à l'étape 1 on trie les intervalles par date de début croissante, alors la méthode est optimale.

On considère $\mathcal{P}$ la partition renvoyée par l'algorithme dans ce cas. On note k son nombre de classes. Pour $\alpha \in [|1, k|]$, on note n_α la taille de la classe α et pour tout $\beta \in [|1, n_\alpha|]$, on note i_α^β le β-ième intervalle de la classe α, quand on les prend par date de début croissante. De plus les classes sont numérotées dans l'ordre de leur création, donc si on considère les i_α^1, ils sont également rangés par ordre croissant de date de début.

Par l'absurde, supposons l'existence de $\mathcal{P}'$ une partition de taille $l < k$, telle que dans chaque classe les intervalles sont deux à deux disjoints.

3. On considère l'intervalle i_{l+1}^1, le premier intervalle de la classe $l + 1$. Lors de son traitement par l'algorithme, il a été mis dans une nouvelle classe. Que peut on alors dire concernant les classes $1, ..., l$?
4. En déduire que l'existence de $\mathcal{P}'$ est absurde.
5. Implémenter cet algorithme en Ocaml. On pourra réutiliser certaines fonctions de l'exercice précédent.