TP : Algorithme glouton et algorithme dynamique

On a ouvert une salle de spectacles : les demandes de réservation affluent ! Dans ce TP, on s'intéresse à la sélection des spectacles programmés dans notre salle, tout d'abord pour maximiser le nombre de spectacles programmés, puis pour maximiser le profit.

I. Maximisation du nombre de spectacles (LISTES)

n personnes veulent organiser des événements dans notre salle avec des horaires de début d_i et de fin $f_i > d_i$ pour chaque événement $i \in [\![0, n-1]\!]$. Les dates d_i et f_i sont par exemple des temps en heures ou en jours depuis un instant de référence. Voici un exemple :

```
# exemple1 ;;
- : (int * int) list = [(8, 10); (1, 2); (6, 10); (3, 7); (1, 3); (4, 5); (5, 7)]
```

Certains horaires sont toutefois incompatibles : les événements associés se superposeraient dans la salle. Par exemple, on ne peut pas programmer l'événement $(3, 7)$ et l'événement $(6, 10)$ dans notre salle. On souhaite maximiser le nombre d'événements programmés, et on appelle solution optimale au problème un choix d'un ensemble de p événements, tels que les intervalles $[d_i, f_i[$ choisis soient compatibles (c'est-à-dire disjoints), avec p maximal. Remarquez que les intervalles sont fermés à gauche et ouvert à droite : on peut programmer les événements $(4, 5)$ et $(5, 7)$ dans notre salle.

On va voir qu'il existe une solution simple pour résoudre le problème, qui suit une approche *gloutonne* : on trie les événements par date de fin f_i croissante, il suffit ensuite de parcourir une seule fois la liste des événements pour choisir ceux que l'on programme dans notre salle.

Question 1. Écrire une fonction permettant de trier une liste de couples (d, f) par date de fin (f) croissante. On utilisera le tri fusion.

```
# tri_fusion exemple1
- : (int * int) list = [(1, 2); (1, 3); (4, 5); (5, 7); (3, 7); (8, 10); (6, 10)]
```

On suppose désormais la liste effectivement triée par date de fin d'événement croissante (on pourra utiliser la liste `exemple1_trie` de l'annexe). L'algorithme utilisé pour programmer un maximum de spectacles dans notre salle consiste à parcourir la liste des événements par date de fin croissante : si l'événement considéré n'est pas incompatible avec ceux précédemment sélectionnés, c'est-à-dire qu'il débute après la fin du dernier choisi, alors on le sélectionne également.

Question 2. Écrire une fonction `programmation q` prenant en entrée une liste d'événements (couples (d, f)) triés par date de fin f croissante, et qui renvoie la liste des spectacles sélectionnés suivant l'algorithme précédent.

```
# programmation exemple1_trie ;;
- : (int * int) list = [(1, 2); (4, 5); (5, 7); (8, 10)]
```

On veut maintenant montrer la correction de l'algorithme : il s'agit de montrer que la liste d'événements sélectionnés est de taille maximale parmi les familles d'événements compatibles.

Question 3. Soit E un ensemble d'événements, dont les éléments sont $e_0, \dots, e_{n-1}$, triés par date de fin croissante.

1. Montrer qu'il existe une solution optimale au problème (i.e un ensemble d'événements compatibles de taille maximale) contenant l'événement e_0.

2. Conclure par récurrence que l'algorithme proposé renvoie bien une solution optimale.

II. Maximisation du profit (TABLEAUX)

Les organisateurs d'événements sont prêts à débourser une certaine somme pour réserver notre salle le temps de leur manifestation. On se pose maintenant comme question la maximisation du profit. L'ensemble des événements est maintenant représenté par un **tableau** de *triplets* de la forme (d_i, f_i, p_i), où :

— les d_i et f_i sont les dates de début et de fin de l'événement numéro i ;

— le poids p_i est le prix qu'est prêt à débourser l'organisateur pour réserver notre salle ;

— les événements sont toujours supposés triés par date de fin f_i croissante.

Par exemple, l'ensemble d'événements peut être le tableau suivant :

```
# exemple2 ;;
- : (int * int * int) array =
[|(1, 2, 100); (1, 3, 150); (4, 5, 250); (3, 7, 400); (5, 7, 150); (8, 10, 200); (6, 10, 500)|]
```

Le calcul d'un ensemble d'événements compatibles maximisant le profit est légèrement plus difficile que celui de la section précédente, mais il reste possible de proposer un algorithme efficace. Pour ce faire, en notant n le nombre d'événements et en supposant ceux-ci indexés de 0 à $n-1$, on note pour $0 \leq i < n$, M_i le profit maximal que l'on peut obtenir sur les événements indexés de 0 à i, c'est-à-dire :

$$M_i = \max \left\{ \sum_{j \in J} p_j \;\middle|\; J \subset [\![0, i]\!], \text{ les événements indicés par } J \text{ sont compatibles} \right\}$$

Question 4. Calculer les valeurs M_i pour $i = 0, \ldots, 6$, avec les événements du tableau `exemple2` donné plus haut.

Dans la suite, on note $i(k)$ le numéro de l'événement qui se termine le plus tard parmi ceux qui se terminent avant le début de l'événement numéro k. Pour `exemple2`, $i(6) = 2$ car l'événement d'indice 2 (c'est-à-dire $(4, 5, 250)$) est celui qui termine le plus tard parmi les événements qui terminent avant la date 6 (début de l'événement numéro 6, qui est $(6, 10, 500)$). On a aussi $i(3) = 1$, car l'événement $(3, 7, 400)$ débute juste après $(1, 3, 150)$.

Fonctions données. Dans la suite, vous pourrez faire usage des fonctions `dd`, `df` et `prix`, prenant toutes trois en entrée un triplet d'entiers, et renvoyant le premier, deuxième et troisième élément du triplet.

Question 5. Écrire une fonction `calculeI t k` prenant en entrée un tel tableau d'événements et un indice k du tableau, et calculant $i(k)$. On renverra -1 s'il n'existe pas d'événement terminant avant le début de l'événement numéro k.

```
# calculeI exemple2 6 ;;
- : int = 2
# calculeI exemple2 3 ;;
- : int = 1
# calculeI exemple2 1 ;;
- : int = -1
```

Question 6. En déduire une fonction `calculetousI t` renvoyant le tableau des $i(k)$ pour $k \in [\![0, n-1]\!]$, avec n la taille de `t`.

```
# calculetousI exemple2 ;;
- : int array = [|-1; -1; 1; 1; 2; 4; 2|]
```

Question 7. Donner la complexité de votre fonction `calculetousI t`.

Question 8. Exhiber une relation entre M_k, M_{k-1} (si $k > 0$), $M_{i(k)}$ (si $i(k) \geq 0$) et p_k, pour tout $k \in [\![0, n-1]\!]$. On distinguera les cas $k = 0$, $i(k) = -1$ et $i(k) \geq 0$. Faites le bien, cette relation est essentielle pour la suite ! Prouvez la relation.

Question 9. Écrire une fonction `calculeM t` prenant en entrée un tableau d'événements et renvoyant le tableau des M_k, pour $k \in [\![0, n-1]\!]$, avec n la taille de `t`. Le calcul des M_k devra s'effectuer itérativement (sans usage de récursivité). Quelle est la complexité de cette fonction ?

Question 10. Écrire une fonction `calculeSol t` prenant en entrée un tableau `t` d'événements et renvoyant **une liste** de numéros d'événements dont la somme des poids vaut M_{n-1}. Vous justifierez l'approche avant de coder.

```
# calculeSol exemple2 ;;
- : int list = [1; 2; 6]
```