TP, Algorithme glouton et algorithme dynamique : Corrigé

I. Maximisation du nombre de spectacles (LISTES)

Question 1. Seule la fonction de fusion diffère de la version « standard ».

```
let rec fission q =match q with
  | [] | [_] -> q,[]
  | x::y::p -> let a,b=fission p in x::a, y::b
;;

let rec fusion p q = match p,q with
  | [],_ | _,[] -> p@q
  | (a,b)::r, (c,d)::s when b<=d -> (a,b)::fusion r q
  | _,y::s -> y::fusion p s
;;

let rec tri_fusion q=match q with
  | [] | [_] -> q
  | _ -> let p1, p2 = fission q in fusion (tri_fusion p1) (tri_fusion p2)
;;
```

Question 2. Il est nécessaire de connaître la date de fin du dernier élément sélectionné lorsqu'on s'intéresse à un nouvel événement. D'où la fonction suivante, faisant usage d'une fonction interne :

```
let programmation q=
  let rec aux t p=match p with
    | [] -> []
    | (d,f)::r -> if d>=t then (d,f)::aux f r else aux t r
 in  let a=List.hd q in a :: aux (snd a) (List.tl q)
;;
```

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. Soit S une solution optimale au problème (i.e un ensemble d'événements compatibles de taille maximale). Supposons que S ne contienne pas e_0. S est non vide (car tout ensemble d'un événement seul est une solution au problème). Considérons l'événement e_i de S avec i minimal. Par définition, tous les autres éléments de S débutent après e_i, et donc après e_0. Ainsi, $S \backslash \{e_i\} \cup \{e_0\}$ est encore une solution au problème, optimale car de même taille que S.
2. Par une récurrence forte sur le nombre d'événements, montrons que l'algorithme renvoie une solution optimale :
   — s'il n'y a aucun événement, l'algorithme est correct !
   — sinon, soit $n \geq 1$ un nombre d'événements. L'algorithme sélectionne l'événement e_0, et cherche une solution optimale T parmi les événements compatibles avec e_0. Par hypothèse de récurrence, il calcule correctement une telle solution, et renvoie $T \cup \{e_0\}$. Or, on a vu que e_0 appartenait à une solution optimale S, et aucun des événements non compatibles avec e_0 ne peut faire partie de S, donc $S \backslash \{e_0\}$ a autant d'éléments que T. L'algorithme est donc correct.
   — par principe de récurrence, il fonctionne.

II. Maximisation du profit (TABLEAUX)

Question 4. Les valeurs sont les suivantes : 100, 150, 400, 550, 550, 750, 900.

Question 5. Par exemple :

```
let calculeI t k =
  let j=ref (k-1) in
  while !j>=0 && df t.( !j) > dd t.(k) do
    decr j
  done ;
  !j
;;
```

Variante avec exception :

```
exception Trouve of int ;;
let calculeI t k =
  try
    for j=k-1 downto 0 do
      if df t.(j) <= dd t.(k) then raise (Trouve j)
    done ;
    -1
  with Trouve x -> x
;;
```

Question 6.

```
let calculetousI t =
  let n=Array.length t in
  let i=Array.make n 0 in
  for j=0 to n-1 do
    i.(j)<-calculeI t j
  done ;
  i
;;
```

Question 7. Clairement en $O(n^2)$.

Question 8. En convenant que $M_{-1} = 0$, on a $M_k = \max(M_{k-1}, p_k + M_{i(k)})$ pour tout k. En effet, $M_k \geq M_{k-1}$ et $M_k \geq p_k + M_{i(k)}$, et donc $M_k \geq \max(M_{k-1}, p_k + M_{i(k)})$, mais l'autre inégalité est claire en discutant si une solution de poids M_k contient ou non l'événement d'indice k.

Question 9. En version courte :

```
let calculeM t =
  let n=Array.length t in
  let m=Array.make n (prix t.(0)) in
  let ci = calculetousI t in
  for i=1 to n-1 do
      m.(i) <- max m.(i-1) (prix t.(i) + if ci.(i) = -1 then 0 else prix t.(ci.(i)))
  done ;
  m
;;
```

Question 10.

```
let calculeSol t =
  let n, c, m  = Array.length t, calculetousI t, calculeM t in
  let q=ref [] in
  let k=ref (n-1) in
  while !k>=0 do
    if !k>0 && m.( !k) = m.( !k-1) then
      decr k
    else (q:= !k :: !q ; k:=c.( !k))
  done ;
  !q
;;
```