CCP 2007. Option Informatique : Algorithmique et Programmation en CaML.

Corrigé pour serveur UPS par JL. Lamard (jean-louis.lamard@prepas.org)

Représentation du problème.

Question 1.

```
let comparer p1 p2 =
  let (a1,b1) = p1 and (a2,b2) = p2 in (a1 = a2) & (b1 = b2)
;;
```

Question 2.

```
let rec taille_aux nb_prov reste = match reste with
  | []         -> nb_prov
  | a :: suite -> taille_aux (succ nb_prov) suite
;;

let taille  = taille_aux 0;;
```

Question 3.

$\det(\overrightarrow{QP},\overrightarrow{QR}) = \|\overrightarrow{QP}\|_2 \times \|\overrightarrow{QR}\|_2 \times \sin(\overrightarrow{QP},\overrightarrow{QR})$ d'où tous les résultats demandés.

Question 4.

```
let croise p q r =
  let (a0,b0) = q and (a1,b1) = p and (a2,b2) = r in
    (a1 - a0) * (b2 - b0) - (a2 - a0) * (b1 - b0)
;;
```

Question 5.

```
let rec separation_aux g d sup_prov inf_prov app_prov reste =
  match reste with
    | [] -> (sup_prov, inf_prov, app_prov)
    | a :: suite -> match ((comparer a g) || (comparer a d)) with
        | true -> separation_aux g d sup_prov inf_prov app_prov suite
        | _    -> let signe = croise d g a in match 0 with
            | _ when (signe > 0) ->
                separation_aux g d (a::sup_prov) inf_prov app_prov suite
            | _ when (signe = 0) ->
                separation_aux g d sup_prov inf_prov (a::app_prov) suite
            | _                  ->
                separation_aux g d sup_prov (a::inf_prov) app_prov suite
;;

let separation p q =
  let (a1,b1) = p and (a2,b2) = q in match 0 with
    | _ when (a2 = a1) -> failwith "Droite verticale !"
    | _ when (a2 > a1) -> separation_aux p q [] [] []
    | _                -> separation_aux q p [] [] []
;;
```

Question 6.

La fonction separation_aux est récursive terminale sur l'ensemble :
point $\times$ point $\times$ suite $\times$ suite $\times$ suite $\times$ suite
avec la graduation $d(g,d,e_1,e_2,e_3,e_4) = taille(e_4)$, la fonction de descente associant à chaque élément un élément de degré diminué d'une unité.
Le nombre d'appel récursif généré par separation e est donc égal à la taille de la liste e.
En outre avant chaque appel une seule opération à temps constant est effectuée (un changement de pointeur de tête de liste). La complexité temporelle est donc linéaire. □

Remarque : comme la récursion est terminale, la complexité spatiale est bornée.

Décomposition en sous-problèmes disjoints.

Question 7.

```
let rec extreme_aux min_prov max_prov reste = match reste with
  | []       -> (min_prov,max_prov)
  | p::suite -> let (a0,b0) = min_prov and (a1,b1) = max_prov and (a,b) = p
                in match (a < a0) with
                  | true -> extreme_aux p max_prov suite
                  | _    -> if (a < a1)
                            then extreme_aux min_prov max_prov suite
                            else extreme_aux min_prov p suite
;;

let extreme e = match e with
  | [] -> failwith "suite vide !"
  | p::suite -> extreme_aux p p suite
;;
```

Question 8.

Supposons par exemple que G ne soit pas un sommet de l'enveloppe convexe. Alors G est un barycentre "positif" des sommets A_i de l'enveloppe convexe ce qui entraîne que x_G est le "même" barycentre des abscisses x_i des sommets A_i. Or $x_G < x_i$ pour tout i (la suite de points est séparable) donc x_G est strictement inférieur à tout barycentre "positif" des x_i. Contradiction.
Si $\mathscr{P}$ est un ensemble de points séparables non vide alors le point le plus à gauche et le point le plus à droite sont des sommets de l'enveloppe convexe. □

Structure d'arbres enveloppants.

Question 9.

La profondeur de a est 0 si $a = \varnothing$ et sinon est égale à $1 + M$ où M est le maximum de la profondeur de $\mathscr{G}(a)$ et de $\mathscr{D}(a)$. □

Question 10.

- La profondeur de l'arbre est clairement maximale et alors égale à n lorsque chaque nœud (sauf le dernier) ne possède qu'un fils non vide.
- Elle est minimale lorsque sur chaque niveau on peut ranger le maximum de points donc lorsque chaque nœud admet 2 fils non vides sauf les feuilles bien sûr et sauf éventuellement les nœud de l'avant-dernier niveau qui peuvent n'avoir qu'un fils.

Soit p la profondeur d'un tel arbre et n son nombre de nœuds.
Numérotons les couches en partant du haut de 0 à $p-1$. Sur la couche k il y a 2^k nœuds si $k < p-1$ et sur la dernière il y en a au moins 1 et au plus 2^{p-1}.
Donc $1+2+2^2+\cdots+2^{p-2} < n \leqslant 1+2+2^2+\cdots+2^{p-1}$ soit $2^{p-1} < n+1 \leqslant 2^p$.
Donc $p-1 < \log_2(n+1) \leqslant p$ donc $p \sim \log_2(n)$.
En fait on a exactement $p = -\operatorname{Int}\big(-\log_2(n+1)\big)$. □

Question 11.

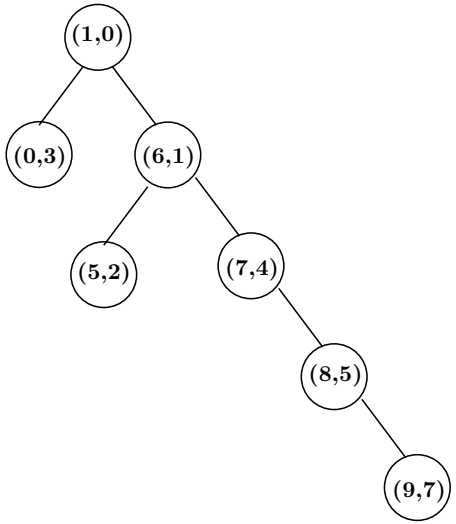
```
let rec profondeur a = match a with
  | Vide -> 0
  | Noeud (fg,p,fd) -> 1 + max (profondeur fg) (profondeur fd)
;;
```

Question 12.

Pour construire l'arbre enveloppant d'une suite de point on remarque que l'étiquette de la racine de l'arbre est le point p_0 le plus bas, que le fils gauche est l'arbre enveloppant de tous les points à gauche de p_0 et que le fils droit est l'arbre enveloppant de tous les points à droite de p_o.

Cette remarque permet de construire l'arbre enveloppant d'une suite de points par récursivité et prouve d'ailleurs son unicité.

Pour la suite de points proposée, on obtient l'arbre suivant :



Question 13.

En considérant que le vide est un arbre enveloppant, en notant x (resp. y) la fonction qui appliquée à un point renvoie son abscisse (resp. son ordonnée) :

$$\mathrm{AEI}(a) \iff a = \varnothing \text{ ou } \begin{cases} \mathrm{AEI}(\mathscr{G}(a)) \\ \mathrm{AEI}(\mathscr{D}(a)) \\ y(\mathscr{E}(a)) < y(\mathscr{E}(\mathscr{G}(a))) \text{ si } \mathscr{G}(a) \neq \varnothing \\ y(\mathscr{E}(a)) < y(\mathscr{E}(\mathscr{D}(a))) \text{ si } \mathscr{D}(a) \neq \varnothing \\ \forall n \in \mathscr{N}(\mathscr{G}(a)) \;\; x(\mathscr{E}(n)) < x(\mathscr{E}(a)) \\ \forall n \in \mathscr{N}(\mathscr{D}(a)) \;\; x(\mathscr{E}(n)) > x(\mathscr{E}(a)) \end{cases}$$

Application au calcul d'une enveloppe convexe.

Question 14.

Remarque : Soit un polygone $(P_1, P_2, \ldots, P_r)$ avec $x_1 < x_{i+1}$ (en notant $P_i = (x_i, y_i)$) et tel que P_i soit en dessous de la droite (P_1P_r) pour i de 2 à $r-1$. Dans la suite un tel polygone sera qualifié d' "inférieur".
Alors un point $M(x,y)$ est sur ou à l'intérieur de ce polygone si et seulement si :

$$\mathscr{S} \iff \begin{cases} M \text{ est en dessous de la droite } (P_1P_r) \\ \exists i \in [1, r-1] \text{ tq } x_i \leqslant x \leqslant x_{i+1} \\ M \text{ est sur ou au-dessus de la droite } (P_iP_{i+1}) \end{cases}$$

Soit a l'arbre enveloppant inférieur construit à partir de $\mathscr{P}$ et soit $A = (a,b)$ le point de $\mathscr{P}$ d'ordonnée minimum (qui est l'étiquette de la racine de a).
Le contour direct est un polygone "inférieur" : $(P_1, P_2, \ldots, P_r)$; en effet par construction $x_i < x_{i+1}$ donc $P_1 = G$, $P_r = D$ et par hypothèse tous les points de $\mathscr{P}$ sauf G et D donc a fortiori tous les points du contour direct sauf P_1 et P_r sont en dessous de la droite (P_1P_r).

Supposons qu'il existe un point $P = (\alpha, \beta)$ de $\mathscr{P}$ à l'extérieur de ce polygone.
Déjà par hypothèse P est sur ou au-dessous de la droite $(GD) = (P_1P_r)$ donc vérifie la première condition de $\mathscr{S}$.
Par ailleurs il vérifie évidemment la seconde puisque P_1 et P_r sont les points les plus à gauche et à droite de $\mathscr{P}$. De manière plus précise il existe i tel que $x_i < \alpha < x_{i+1}$ (séparabilité).
Alors la troisième condition n'est pas vérifiée donc P est en dessous de la droite (P_iP_{i+1}).
Or puisque A est un point du contour direct, $x_{i+1} \leqslant a$ ou $a \leqslant x_i$ Plaçons nous par exemple dans le premier cas.
Cela signifie que P_i et P_{i+1} correspondent à des noeuds de la branche la plus à gauche de l'arbre a donc P_i est le fils gauche de P_{i+1} et la pente de la droite (P_iP_{i+1}) est négative.
Comme $x_i < \alpha < x_{i+1}$ le fait que P soit au-dessous de (P_iP_{i+1}) implique $\beta < y_i$ car la pente de (P_iP) est inférieure à celle de (P_iP_{i+1}) donc négative.
Ainsi P est un point à gauche de P_{i+1} dont l'ordonnée est strictement inférieure à celle de P_i.
Contradiction avec le fait que P_i soit le fils gauche de P_{i+1}.

Si tous les points d'un ensemble de points se trouvent dessous la droite joignant le point le plus à gauche et le point le plus à droite alors le contour direct de l'arbre enveloppant inférieur construit à partir de cet ensemble définit un polygone tels que tous les points de l'ensemble soient à l'intérieur ou sur ce polygone. □

Question 15.

On commence par écrire des fonctions qui renvoie la liste des étiquettes des noeuds de la branche la plus à gauche et de la branche la plus à droite parcourues dans le sens haut-bas. Ces fonctions ne parcourent respectivement que la branche la plus à gauche et la plus à droite :

```
let rec developper_gauche a = match a with
  | Vide -> []
  | Noeud(fg, p, fd) -> p :: (developper_gauche fg)
;;

let rec developper_droite a = match a with
  | Vide -> []
  | Noeud(fg, p, fd) -> p :: (developper_droite fd)
;;

let developper a = (rev (developper_gauche a)) @ tl(developper_droite a);;
```

Question 16.

L'enveloppe convexe de $\mathscr{C}$ contient $\mathscr{P}$ d'après la question 14 : $\mathscr{P} \subset \mathscr{E}_C(\mathscr{C})$.
Donc par isotonie de l'application "enveloppe convexe" : $\mathscr{E}_C(\mathscr{P}) \subset \mathscr{E}_C\Big(\mathscr{E}_C(\mathscr{C})\Big) = \mathscr{E}_C(\mathscr{C})$
Par ailleurs on a puisque $\mathscr{C} \subset \mathscr{P}$ l'inclusion $\mathscr{E}_C(\mathscr{C}) \subset \mathscr{E}_C(\mathscr{P})$.
Ainsi $\mathscr{E}_C(\mathscr{C}) = \mathscr{E}_C(\mathscr{P})$ ce qui prouve que les sommets de l'enveloppe convexe de $\mathscr{P}$ appartiennent à $\mathscr{C}$. □

Question 17.

Supposons $[p_n, p_{n+1}, p_{n+2}] \geqslant 0$. D'après la question 3, cela signifie que p_{n+1} est sur ou au-dessus de la droite $(p_n p_{n+2})$.
Or le polygone $\mathscr{C}'$ obtenu à partir de $\mathscr{C}$ en retirant le point p_{n+1} reste "inférieur" car abscisses restent bien croissantes et les points les plus à gauche et à droite n'ont pas changé.
Il découle alors de la remarque liminaire de la question 14 que le point p_{n+1} est intérieur ou sur le polygone $\mathscr{C}'$.
Donc l'enveloppe convexe de $\mathscr{C}'$ contient $\mathscr{C}$. La réciproque est évidente puisque $\mathscr{C}' \subset \mathscr{C}$.
Ainsi $\mathscr{E}_C(\mathscr{P}) = \mathscr{E}_C(\mathscr{C}) = \mathscr{E}_C(\mathscr{C}')$ ce qui prouve que p_{n+1} n'est pas un sommet de $\mathscr{E}_C(\mathscr{P})$. □

Question 18.

La sous-suite maximale $\mathscr{C}_{max}$ en question est la sous-suite de $\mathscr{C}$ obtenue en retirant tous les points du type du point p_{n+1} de la question précédente.
D'après la question précédente on a donc $\mathscr{E}_C(\mathscr{C}_{max}) = \mathscr{E}_C(\mathscr{C}) = \mathscr{E}_C(\mathscr{P})$.
Pour prouver qu'il s'agit bien de la suite des sommets de l'enveloppe convexe, il reste à prouver que si l'on enlève un point de cette suite notée $(p_1, p_2, \ldots, p_r)$ alors l'enveloppe convexe change.
On a $p_1 = g$ et $p_r = d$ donc si l'on retire p_1 par exemple, p_1 ne sera plus dans l'enveloppe convexe.
Soit p_{n+1} avec $1 < n+1 < r$. Alors $[p_n, p_{n+1}, p_{n+2}] < 0$ ce qui signifie que p_{n+1} est en-dessous de la droite $(p_n p_{n+2})$ donc que p_{n+1} est extérieur au polygone obtenu en retirant p_{n+1} d'après la remarque liminaire de la question 14 (en effet ce polygone reste bien "inférieur").

La suite $\mathscr{C}_{max}$ est bien égale à la suite des sommets de $\mathscr{E}_C(\mathscr{P})$. □

Question 19.

```
let rec filtrer s = match s with
  | []     -> s
  | [a]    -> s
  | [a; b] -> s
  | a::b::c::suite -> let signe = (croise a b c) in match signe with
      | _ when (signe >=0) -> filtrer (a::c::suite)
      | _                  -> a::(filtrer (b::c::suite))
;;
```

Opérations sur un arbre enveloppant inférieur.

Question 20.

```
let rec contenir p a = match a with
  | Vide -> false
  | Noeud(fg,m,fd) -> let (x,y) = p and (a,b) = m in match 0 with
      | _ when x = a -> y = b
      | _ when x > a -> contenir p fd
      | _            -> contenir p fg
;;
```

Question 21.

L'algorithme ci-dessus parcourt au plus une branche.
Dans le meilleur des cas p est l'étiquette de la racine de l'arbre et alors aucun appel récursif.
Le pire des cas est celui où l'algorithme parcourt la branche la plus longue toute entière sans succès. Le nombre d'appel récursif est alors égal à la profondeur de l'arbre.
Cette complexité peut atteindre n avec un arbre dont par exemple tous les nœuds n'ont qu'un fils droit i.e. pour un ensemble de points $(p_1, p_2, \ldots, p_n)$ tel que $x_i < x_{i+1}$ et $y_i < y_{i+1}$ pour la recherche d'un point $p = (x, y)$ tel que $x_n < x$. □

Question 22.

Le programme suivant renvoie l'arbre enveloppant extrait de a dont les points ont une abscisse inférieure ou égale à x et celui dont les points ont une abscisse strictement supérieure.

```
let rec scinder x a = match a with
  | Vide           -> (Vide,Vide)
  | Noeud(fg,p,fd) -> let (a,b) = p in match x < a with
      | true -> let (a1,a2) = scinder x fg in (a1, Noeud(a2,p,fd))
      | _    -> let (a1,a2) = scinder x fd in (Noeud(fg,p,a1), a2)
;;
```

Question 23.

```
let rec ajouter p a = let (x,y) = p in match a with
  | Vide -> Noeud(Vide,p,Vide)
  | Noeud(fg,m,fd) -> let (r,s) = m in match y < s with
      | true -> let (a1,a2) = scinder x a in Noeud(a1,p,a2)
      | _    -> if x < r then Noeud(ajouter p fg,m,fd)
                         else Noeud(fg,m,ajouter p fd)
;;
```

Question 24.

- Le meilleur cas est celui où l'arbre enveloppant ne comporte qu'une branche et que le point à insérer se place en haut de l'arbre. Par exemple lorsque les points p_i correspondants aux nœuds de l'arbre sont tels que $x_i < x_{i+1}$ et $y_i < y_{i+1}$ et que le point p à insérer vérifie $x < x_1$ et $y < y_1$.
  Il n'y a alors aucun appel récursif pour la fonction `ajouter` et un seul pour la fonction `scinder` qui est appelée une fois par `ajouter`.
- Le pire cas est celui où l'arbre a la même forme mais où le point à insérer vient en bas de l'arbre. Dans notre exemple cela correspond au cas où $x > x_n$ et $y > y_n$.
  Dans ce cas il y a n appels récursifs à la fonction ajouter.

FIN