Enveloppe Convexe - Correction

Samy Jaziri

1 Introduction

Structure de donnée

Ce sujet s'attaque au calcul de l'enveloppe convexe d'un ensemble de points par un algorithme *diviser pour régner*. On travaille en deux dimensions, avec des points de coordonnées entières.

```
type point = int * int ;;
```

Il s'agit ensuite de représenter un ensemble de point. Deux choix s'offrent à nous : listes et vecteurs. On doit effectuer les opérations suivantes sur l'ensemble :

- Parcours dans les deux sens,
- Tri,
- Division en deux sous-ensembles,
- Fusion de deux sous-ensembles.

L'algorithme de tri rapide peut être implémentés en place si on choisit des vecteurs. Ce qui nous permettra d'économiser du temps et de l'espace. De plus, on devra réaliser des parcours aisément dans les deux sens, ce qui constitue un argument supplémentaire pour l'utilisation de vecteurs.

```
type ensemble = point array ;;
```

Tri rapide Générique

L'énoncé demandait un tri rapide prenant une fonction de comparaison quelconque. On propose ici une implémentation générique pour des ensembles quelconque représentés par un vecteur.

```
type ordre = PLUS_PETIT | PLUS_GRAND | EGAUX

let echanger ( vecteur : 'a array ) ( i : int ) ( j : int ) : unit =
  let tampon = vecteur.(i) in
  vecteur.(i) <- vecteur.(j) ;
  vecteur.(j) <- tampon ;;

let partition
    ( vecteur : 'a array )
    ( debut : int )
    ( fin : int )
    ( ordre : 'a -> 'a -> ordre )
  : int =
  let pivot = vecteur.(debut) in
  let limite = ref debut in
  for i = debut + 1 to fin do
    if ordre vecteur.(i) pivot == PLUS_PETIT then (
     limite := !limite + 1 ;
     echanger vecteur i !limite ;
    )
  done ;
  echanger vecteur debut !limite ;
  !limite ;;

let tri_rapide
      ( vecteur : 'a array ) ( comparaison : 'a -> 'a -> ordre ) : unit =
  let rec tri_rapide_en_place ( debut : int ) ( fin : int ) : unit =
    if debut < fin - 1 then (
      let limite = partition vecteur debut fin comparaison in
      tri_rapide_en_place debut limite ;
      tri_rapide_en_place ( limite + 1 ) fin ;
    ) in
  tri_rapide_en_place 0 ( ( Array.length vecteur ) - 1 ) ;;
```

On se tournera vers le cours pour la preuve de correction du tri rapide.

Génération d'ensembles aléatoires

Pas de difficulté ici

```
(*
Paramètre : n entier positif
Génère u_n tel que décrit dans l'énoncé
*)
let rec u ( n : int ) : int =
  if n == 0 then u0 else 37698 * ( u ( n - 1) ) mod 524287 ;;
```

On construit les ensembles E_n comme décrit dans l'énoncé.

```
(*
Paramètre = n : entier positif
Génère E_n tel que décrit dans l'énoncé
*)
let rec e ( n : int ) : ensemble =
  let ensemble = Array.make n ( 0, 0 ) in
  for i = 0 to n - 1 do
    ensemble.(i) <- ( u ( 2 * i ), u ( 2 * i + 1 ) )
  done ;
  ensemble ;;
```

Tri des sous-ensembles de $\mathbb{N}^2$

On implémente l'ordre lexicographique.

```
(*
  Paramètre = point_g, point_d : deux points de N^2
  Renvoie le résultat de la comparaison entre point_g et point_d selon l'ordre
  lexicographique.
*)
let ordre_lex ( point_g : point ) ( point_d : point ) : ordre =
  let xg, yg = point_g in
  let xd, yd = point_d in
  (*
    Paramtres n, m : deux entiers
    Renvoie le résultat de la comparaison de n et m dans le type ordre.
  *)
  let ordre_int ( n : int ) ( m :int ) : ordre =
    if n < m then PLUS_PETIT
    else if n > m then PLUS_GRAND
    else EGAUX in
  let ordre_x = ordre_int xg xd in
  match ordre_x with
  | EGAUX -> ordre_int yg yd
  | _ -> ordre_x ;;
```

Un simple algorithme de max pour le résultat en utilisant l'ordre lexicographique comme comparaison. On note pour plus tard que l'indice renvoyé dans le cas où le maximum apparaît plusieurs fois est le premier indice rencontré.

```
(*
  Paramètre = ensemble : un ensemble non vide de N^2
  Renvoie le plus grand indice où apparaît le plus grand point de l'ensemble.
  Pour l'ordre lexicographique.
*)
let max_ensemble ( ensemble : ensemble ) : int =
  let ipoint_max = ref 0 in
  let cardinal = Array.length ensemble in
  for i = 1 to cardinal - 1 do
    if ordre_lex ensemble.(i) ensemble.(!ipoint_max) == PLUS_GRAND
    then ipoint_max := i
  done ;
  !ipoint_max ;;
```

2 Calcul de l'enveloppe convexe d'un sous-ensemble de $\mathbb{N}^2$

Principe général

L'enveloppe d'un singleton $\{p\}$ sera représenté par le vecteur `[| p; p |]` puisqu'on duplique le premier point de l'enveloppe.

```
(*
  Paramètre = ensemble : un sous-ensemble de N^2
  Renvoie l'enveloppe convexe de l'ensemble comme décrite dans l'énoncé
*)
let enveloppe_convexe ( ensemble : ensemble ) : ensemble =
  (* On trie l'ensemble pour effectuer les divisions succesive *)
  tri_rapide ensemble ordre_lex ;
  (* On gagne de l'espace en réalisant les divisions en place *)
  let rec enveloppe_avec_division_en_place
            ( ensemble : ensemble ) ( debut : int ) ( fin : int ) : ensemble =
    if debut == fin (* Enveloppe convexe d'un singleton *)
    then [| ensemble.(debut) ; ensemble.(debut) |]
    else ( (* Fusion des enveloppes du sous-ensemble droit et gauche *)
      fusion_enveloppes
        ( enveloppe_avec_division_en_place
            ensemble debut ( ( debut + fin ) / 2 ) )
        ( enveloppe_avec_division_en_place
            ensemble ( ( debut + fin ) / 2 + 1 ) fin )
    ) in
  enveloppe_avec_division_en_place
    ensemble 0 ( ( Array.length ensemble ) - 1 ) ;;
```

Complexité : L'algorithme commence par un tri rapide en $\mathcal{O}(n^2)$. On note $\mathcal{C}(n)$ la complexité de `enveloppe_avec_division_en_place` sur un ensemble de taille n ($\texttt{fin} - \texttt{debut} + \texttt{1} = \texttt{n}$). En

remarquant que le cardinal de l'enveloppe convexe d'un ensemble est plus petit que le cardinal de l'ensemble initial on peut donc écrire :

$$\mathcal{C}(n) = 2 * \mathcal{C}(\frac{n}{2}) + \mathcal{O}(n)$$

D'après le théorème maître `enveloppe_avec_division_en_place` a donc une complexité en $\mathcal{O}(n \log(n))$. La complexité finale est au pire en $\mathcal{O}(n^2)$. On remarque d'une part que la complexité en moyenne est en $\mathcal{O}(n \log(n))$ puisqu'en moyenne le tri rapide a une complexité de $\mathcal{O}(n \log(n))$. On remarque aussi qu'on peut faire baisser la complexité au pire de l'algorithme à $\mathcal{O}(n \log(n))$ en utilisant un tri fusion au lieu d'un truc rapide.

Recherche de tangentes

La fonction `direction` utilise la formule du produit vectoriel donnée par l'énoncé.

```
type sens = TRIGO | ANTI_TRIGO | ALIGNE ;;

(*
  Paramètre = point_1, point_2 : deux points
  Paramètre = point : un troisième point.
  Renvoie le sens de rangement de point_1, point_2 et point.
*)

let direction
    ( point_1 : point )
    ( point_2 : point )
    ( point : point )
  : sens =
  let x1, y1 = point_1 in
  let x2, y2 = point_2 in
  let x, y = point in
  let z_produit = ( x1 - x2 ) * ( y - y2 ) - ( y1 - y2 ) * ( x - x2 ) in
  if z_produit < 0 then TRIGO
  else if z_produit > 0 then ANTI_TRIGO
  else ALIGNE ;;
```

On utilisera cette fonction qui donne le cardinal de l'enveloppe, c'est-à-dire le cardinal de l'ensemble la représentant auquel on enlève 1 pour le premier point qui est dupliqué à la fin.

```
let cardinal_enveloppe ( enveloppe : ensemble ) : int =
  ( Array.length enveloppe ) - 1 ;;
```

On suit scrupuleusement la description de l'algorithme de l'énoncé. On choisit d'utiliser une fonction récursive plutôt qu'une boucle `while`.

```
let rec tangente_haute
    ( enveloppe_g : ensemble )
    ( enveloppe_d : ensemble )
    ( i_point_g : int )
    ( i_point_d : int )
    : int * int =
  let point_g = enveloppe_g.( i_point_g ) in
  let point_d = enveloppe_d.( i_point_d ) in
  (* Etape (1) *)
  if i_point_g < ( cardinal_enveloppe enveloppe_g )
     && ( direction point_d point_g enveloppe_g.( i_point_g + 1 )
          == ANTI_TRIGO )
  then tangente_haute enveloppe_g enveloppe_d ( i_point_g + 1 ) ( i_point_d )
  (* Etape (2) *)
  else if i_point_d > 0
     && ( direction point_g point_d enveloppe_d.( i_point_d - 1 ) == TRIGO )
  then tangente_haute enveloppe_g enveloppe_d ( i_point_g ) ( i_point_d - 1 )
  (* Condition de terminaison attente *)
  else ( i_point_g, i_point_d ) ;;

let rec tangente_basse
    ( enveloppe_g : ensemble )
    ( enveloppe_d : ensemble )
    ( i_point_g : int )
    ( i_point_d : int )
    : int * int =
  let point_g = enveloppe_g.( i_point_g ) in
  let point_d = enveloppe_d.( i_point_d ) in
  (* Etape (3) *)
  if i_point_g > 0
     && ( direction point_d point_g enveloppe_g.( i_point_g - 1 ) == TRIGO )
  then tangente_basse enveloppe_g enveloppe_d ( i_point_g - 1 ) ( i_point_d )
  (* Etape (4) *)
  else if i_point_d < ( cardinal_enveloppe enveloppe_d )
     && ( direction point_g point_d enveloppe_d.( i_point_d + 1 )
          == ANTI_TRIGO )
  then tangente_basse enveloppe_g enveloppe_d ( i_point_g ) ( i_point_d + 1 )
  (* Condition de terminaison attente *)
  else ( i_point_g, i_point_d ) ;;
```

On termine par l'algorithme de fusion. Les cas dégénérés sont les suivants :

- Si p_h et p_b on les mêmes indices. On déplace p_h dans le sens trigonométrique. Tout d'abord le déplacement est possible : en effet si l'indice de p_h est le dernier de l'ensemble, c'est que p_h est en réalité la copie du premier point de l'enveloppe. Comme l'indice de p_b décroît après son initialisation dans l'algorithme, si l'indice de p_b est égal à celu ide p_h donc au dernier point de l'enveloppe, c'est que p_b a été initialisé à ce point-là. p_b est initialisé à `( max_ensemble`

`enveloppe_g`) par conséquent le dernier point de l'enveloppe est aussi le plus grand. Or ce point apparaît deux fois dans l'ensemble, au début et à la fin et `max_ensemble` renvoie le premier indice auquel le point maximal apparaît. On obtient une absurdité, ainsi p_h n'est pas le dernier indice de l'enveloppe. Ensuite on montre que l'on peut faire ce déplacement sans impacter la correction de l'algorithme. L'énoncé nous dit que ce cas correspond à une enveloppe d'un ensemble à une dimension (ensemble plat ou singleton). Dans ces deux cas qu'importe de savoir si le point p_h sera ajouté avant q_b ou après q_h, tant qu'il n'est ajouté qu'une seule fois.

- Si q_h est la copie du premier point de l'ensemble et q_b est le premier point de l'ensemble. On déplace q_h dans le sens anti-trigonométrique. Tout d'abord le déplacement est possible, car une enveloppe est représentée par un vecteur d'au moins 2 éléments. Ensuite on montre que l'on peut faire ce déplacement sans impacter la correction de l'algorithme. L'énoncé nous dit que ce cas correspond à une enveloppe d'un ensemble à une dimension (ensemble plat ou singleton). Dans ces deux cas qu'importe de savoir si le point q_h sera ajouté après p_b ou juste avant p_h, tant qu'il n'est ajouté qu'une seule fois.

Une fois p_h, q_h, p_b et q_b calculés on peut obtenir le cardinal de l'enveloppe en additionnant les points restant dans F_1 (i.e. le cardinal de F_1 moins le nombre de point entre p_b et p_g exclus) et les points restant dans F_2 (i.e. le nombre de point entre q_b et q_h inclus). On alloue un `vecteur` de taille : le cardinal de l'enveloppe plus 1.
Pour remplir le vecteur :

- on part du premier point de F_1 qui est nécessairement dans l'enveloppe finale au regard de l'algorithme et qui sera par définition le point le plus petit de $F_1 \cup F_2$
- on parcourt les point à ajouter dans le sens trigonométrique :
  - on parcourt F_1 jusqu'à p_b.
  - on reprend à q_b et on parcours F_2 jusqu'à q_h
  - on reprend à p_h et on parcours F_1 jusqu'au dernier indice, copie du premier point, nécessairement dans l'enveloppe finale au regard de l'algorithme.

Ainsi les points sont alignés ou rangés dans le sens trigonométrique (d'après la définition de p_h, q_h, p_b et q_b), le premier point est bien le plus petit pour l'ordre lexicographique et le dernier est une copie du premier.

```
let fusion_enveloppes
      ( enveloppe_g : ensemble ) ( enveloppe_d : ensemble ) : ensemble =
  let i_point_max_g = max_ensemble enveloppe_g in
  let cardinal_enveloppe_d = cardinal_enveloppe enveloppe_d in
  let cardinal_enveloppe_g = cardinal_enveloppe enveloppe_g in
  (* On fait l'initialisation comme décrite dans l'algorithme *)
  let i_point_hg, i_point_hd =
    tangente_haute enveloppe_g enveloppe_d i_point_max_g cardinal_enveloppe_d in
  let i_point_bg, i_point_bd =
    tangente_basse enveloppe_g enveloppe_d i_point_max_g 0 in

  (*** Cas dégénérés : ***)
  let i_point_hg =
    if i_point_bg == i_point_hg then i_point_hg + 1 else i_point_hg in
  let i_point_hd =
    if i_point_hd == cardinal_enveloppe_d && i_point_bd == 0
    then i_point_hd - 1 else i_point_hd in
  (*** Construction de l'enveloppe ***)
  let cardinal_enveloppe = cardinal_enveloppe_g + i_point_bg - i_point_hg + 2
                         + i_point_hd - i_point_bd + 1 in
  let enveloppe = Array.make cardinal_enveloppe ( 0, 0 ) in
  (*** Remplissage de l'enveloppe ***)
  let indice_enveloppe = ref 0 in
  for i = 0 to i_point_bg do
      enveloppe.( !indice_enveloppe ) <- enveloppe_g.(i) ;
      indice_enveloppe := !indice_enveloppe + 1 ;
  done ;
  for i = i_point_bd to i_point_hd do
      enveloppe.( !indice_enveloppe ) <- enveloppe_d.(i) ;
      indice_enveloppe := !indice_enveloppe + 1 ;
  done ;
  for i = i_point_hg to cardinal_enveloppe_g do
      enveloppe.( !indice_enveloppe ) <- enveloppe_g.(i) ;
      indice_enveloppe := !indice_enveloppe + 1 ;
  done ;
  enveloppe ;;
```

3 Affichage des résultats de la fiche réponse

```
let sprint_point ( p : point ) : string =
  let x, y = p in Printf.sprintf "( %i, %i )" x y ;;

Printf.printf "Question 3 : a) %i  b) %i  c) %i  \n"
  ( u 30 ) ( u 60 ) ( u 90 ) ;;

let e40 = e 40 ;;
let e270 = e 270 ;;
let e500 = e 500 ;;

let smax40 = sprint_point e40.( max_ensemble e40 ) in
let smax270 = sprint_point e270.( max_ensemble e270 ) in
let smax500 = sprint_point e500.( max_ensemble e500 ) in
Printf.printf
  "Question 4 : a) %s  b) %s  c) %s  \n" smax40 smax270 smax500 ;;

tri_rapide e40 ordre_lex ;;
tri_rapide e270 ordre_lex ;;
tri_rapide e500 ordre_lex ;;

let sp40 = sprint_point e40.(21) in
let sp270 = sprint_point e270.(144) in
let sp500 = sprint_point e500.(212) in
Printf.printf
  "Question 5 : a) %s  b) %s  c) %s \n" sp40 sp270 sp500;;

let f100 = enveloppe_convexe ( e 100 ) ;;
let f212 = enveloppe_convexe ( e 212 ) ;;
let f550 = enveloppe_convexe ( e 550 ) ;;

let car100 = cardinal_enveloppe f100 in
let car212 = cardinal_enveloppe f212 in
let car550 = cardinal_enveloppe f550 in
Printf.printf "Question 9 : a) %i  b) %i  c) %i \n" car100 car212 car550 ;;
```