Concours blanc

Option informatique, première année

Julien Reichert

Toutes les fonctions de ce devoir sont à écrire en Ocaml. Si la transcription en Ocaml d'un algorithme pose problème, il est cependant envisageable de l'écrire en pseudo-code.

On se place pour l'ensemble du sujet dans $\mathbb{R}^2$, et on représentera les nombres par des flottants.

1 Un peu de géométrie

Certaines des fonctions demandées dans cette section peuvent être réutilisées dans la section suivante. On peut à tout moment admettre l'existence d'une fonction provenant d'une question antérieure, même si la question n'a pas été traitée.

Exercice 1 : Écrire une fonction `pente (x1, y1) (x2, y2)` calculant la pente du segment allant du point `(x1, y1)` au point `(x2, y2)` (l'ordre est important). En déduire une fonction `angle` prenant en entrée les mêmes arguments et calculant l'angle orienté entre le vecteur de coordonnées $(1, 0)$ et le vecteur du premier point au second.

On signale l'existence de fonctions trigonométriques en Ocaml, utilisant la convention anglaise et toutes de signature `float -> float` : `cos`, `sin`, `tan`, `acos`, `asin`, `atan`. Les angles sont exprimés en radians.

Exercice 2 : Écrire une fonction `position (x, y) ((x1, y1), (x2, y2))` déterminant si le point de coordonnées `(x, y)` est au-dessus de la droite passant par les points de coordonnées `(x1, y1)` et `(x2, y2)` (auquel cas la fonction renvoie `1`) ou au-dessous de cette droite (auquel cas la fonction renvoie `-1`) ou sur la droite (auquel cas la fonction renvoie `0`). Si la droite est verticale, on remplace « au-dessus » par « à gauche ».

Exercice 3 : Écrire une fonction `hitbox_cc ((x1, y1), r1) ((x2, y2), r2)` déterminant si les disques fermés de centres respectifs `(x1, y1)` et `(x2, y2)` et de rayons respectifs `r1` et `r2` s'intersectent.

Exercice 4 : Écrire une fonction `hitbox_rr ((x1, y1),(x2, y2)) ((x3, y3),(x4, y4))` déterminant si les rectangles pleins de coins en bas à gauche respectifs `(x1, y1)` et `(x3, y3)` et de coins en haut à droite respectifs `(x2, y2)` et `(x4, y4)` s'intersectent.

Exercice 5 (vraiment difficile) : Écrire une fonction `intersection ((x1, y1),(x2, y2)) ((x3, y3),(x4, y4))` et déterminant si les segments formés par les deux premiers points et par les deux derniers points s'intersectent.

2 Calcul d'enveloppe convexe

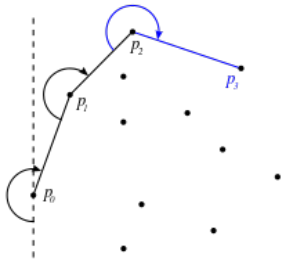
Soit un ensemble E de points du plan. L'enveloppe convexe de E est le plus petit ensemble convexe contenant E, c'est-à-dire un ensemble $X \supset E$ tel que pour tout couple $(x, y) \in X^2$, le segment $[x, y]$ est totalement inclus dans X. Il s'agit alors d'un polygone dont l'ensemble des sommets est un $S \subset E$, et le calcul de l'enveloppe convexe de E revient à déterminer d'une part quels points de E sont dans S et d'autre part quelles sont les arêtes du polygone.

On pourra supposer sans vérification tous les points deux à deux distincts dans les fonctions.

Algorithme de Jarvis (dit du paquet cadeau)

Le principe de ce premier algorithme est simple : un point extrémal (le plus à gauche, disons) est utilisé comme point initial car il est forcément un sommet de l'enveloppe convexe ou sur une de ses arêtes ; on est d'ailleurs certain qu'il en est un sommet s'il est de plus par exemple le plus haut parmi les points ayant l'abscisse minimale. Ensuite, on va « emballer » l'ensemble des points en prenant pour point suivant celui qui forme le plus petit angle (dans le sens horaire) par rapport à la verticale vers le haut avec le point de départ (*), et qui est alors nécessairement sur l'enveloppe convexe, puis à chaque fois on prend pour point suivant celui qui forme le plus petit angle par rapport au segment reliant l'avant-dernier point retenu et le dernier point retenu (l'ordre est important) (*), en prenant le point le plus éloigné en cas d'égalité pour ne pas avoir deux arêtes alignées (en particulier l'angle ne pourra pas être nul). L'algorithme s'arrête une fois le point de départ rejoint, et l'enveloppe convexe est la liste des points retenus dans l'ordre.

(*) C'est aussi le point P' tel qu'en notant P le dernier point retenu tous les autres points sont à droite du vecteur de P à P' ou alignés mais plus proches de P.



Exercice 6 : Écrire cet algorithme en Ocaml. On pourra décomposer les étapes ainsi...

Exercice 6a : Écrire une fonction `extreme_gauche l` qui prend en entrée une liste de couples (représentant des points) et qui retourne le couple de premier élément minimal et, en cas d'égalité, de second élément maximal (pour la cohérence).

Exercice 6b : Écrire une fonction `prochain (pt1, pt2) l` qui prend en entrée un couple de couples et une liste de couples et qui retourne le couple `pt` de la liste tel que le vecteur de `pt2` à `pt` forme un angle minimal avec le vecteur allant de `pt1` à `pt2` (dans le sens horaire, pour la cohérence), et tel que `pt` soit plus éloigné de `pt2` que tout point donnant le même angle.

Exercice 6b bis : ... **OU** (c'est ce que j'ai fait) écrire une fonction `prochain pt1 l` qui prend en entrée un couple et une liste de couples et qui retourne le couple `pt` de la liste tel que tous les autres points soient à droite du vecteur de `pt1` à `pt`, et tel que `pt` soit plus éloigné de `pt1` que tout point aligné avec ces deux points.

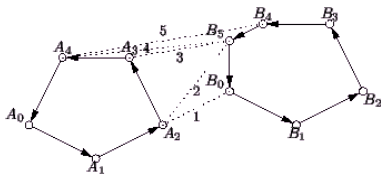
Exercice 6c : Rassembler le tout en une fonction `cadeau points`.

Exercice 7 : Prouver la terminaison du programme (indice : si c'est trivial, c'est que l'implémentation est ratée).

Exercice 8 : Déterminer sa complexité dans le pire des cas.

Algorithme diviser pour régner

Pour trouver l'enveloppe convexe d'un ensemble de points, on peut utiliser une solution astucieuse : cet ensemble peut être séparé en deux : les points les plus à gauche et les points les plus à droite. Alors l'enveloppe convexe de l'ensemble total se déduit des enveloppes convexes des deux ensembles restreints en raccordant deux fois deux points, un de chaque ensemble, pour former une sorte de plafond et de plancher, ce qui remplacera quelques arêtes des deux enveloppes convexes. Pour déterminer les points reliés de chaque ensemble, il s'agit de déterminer, en étudiant chaque point un nombre constant de fois pour des raisons de complexité, par quel couple de points passe une droite séparant le plan en deux zones telles qu'une de ces zones contienne tous les points des deux enveloppes convexes. Le principe pour déterminer le plafond est alors le suivant : on part du point le plus à droite de la zone de gauche et du point le plus à gauche de la zone de droite, et on parcourt l'enveloppe de gauche dans le sens trigonométrique et l'enveloppe de droite dans le sens horaire, en s'arrêtant dès que le point suivant de chaque enveloppe serait sous la droite formée par les points actuels. Le plafond est alors le segment reliant les deux points retenus en fin de compte.



Pour obtenir le plancher, on fait un traitement analogue.

Exercice 9 : Écrire cet algorithme en Ocaml. On pourra décomposer les étapes ainsi...

Non-exercice 9a : On signale l'existence de deux fonctions `List.sort : ('a -> 'a -> int) -> 'a list -> 'a list` et `Array.sort : ('a -> 'a -> int) -> 'a array -> unit` prenant en arguments une fonction adaptée aux comparaisons (par exemple la fonction `compare`) et une liste ou un tableau, respectivement ; la première renvoie une version croissante au sens de la fonction fournie de la liste, et la deuxième trie en place le tableau selon les mêmes conditions.

Pour qu'une fonction `f` soit adaptée aux comparaisons, il faut que pour tout `x`, tout `y` et tout `z` du type des arguments de `f`, on ait `f x y` de signe opposé à `f y x`, si `f x y` et `f y z` sont de même signe alors `f x z` est aussi du même signe et pour la pertinence du tri `f x y` nul si et seulement si `x` et `y` sont égaux.
Ainsi, si on fait `Array.sort f t`, après cet appel on aura, pour tout couple `(i, j)` d'indices autorisés dans `t` vérifiant `i < j`, `f t.(i) t.(j) <= 0`.

La complexité en temps des fonctions de tri sont en $\mathcal{O}(n \log n)$ appels à la fonction de comparaison et en $\mathcal{O}(n \log n)$ affectations, où n est la taille de l'entrée.

Avant de commencer, mieux vaut décider au vu de tout ce qu'il y a à faire dans l'exercice si les points seront mis dans une liste (choix que j'ai retenu) ou un tableau. En pratique, réfléchir à la structure de données doit être la première étape dans tous les travaux de programmation.

Exercice 9a : Écrire une fonction de comparaison utile pour trier les points par abscisse croissante (gestion libre des égalités d'abscisses).

Exercice 9b : Quel code va-t-on écrire pour faire ce tri ? Où va-t-il se placer par rapport à la boucle principale / à l'appel récursif initial ?

Exercice 9c : Écrire une fonction qui prend en entrée une liste et qui renvoie un couple de listes : la première moitié et la deuxième moitié de l'argument. Il suffit que les tailles diffèrent d'un au plus et il faut que toutes les utilisations de cette fonction soient cohérentes. Bien entendu, si la représentation choisie pour cet exercice est le tableau, cette

fonction n'aura pas à être utilisée, mais il faut l'écrire tout de même.

Exercice 9d : Écrire une fonction `plafond gauche droite` qui prend en entrée deux listes ou tableaux de points représentant deux enveloppes convexes et qui retourne le plafond de l'enveloppe convexe globale, en tant que couple de points. Les enveloppes convexes seront supposées triées de sorte que le premier point est au choix du programmeur, mais chaque point suivant est relié au précédent par une arête de l'enveloppe convexe, quel que soit le sens de parcours (rester cohérent, là aussi). Le dernier point peut être une répétition du premier si besoin.

On peut admettre que `plancher`, analogue à `plafond`, est écrite à ce stade. Donner tout de même sa spécification précise dans ce cas, notamment les contraintes sur les arguments, sur leur ordre, et la signification du couple en sortie.

Exercice 9e : Écrire une fonction qui prend en entrée deux enveloppes convexes et qui renvoie l'enveloppe convexe fusionnée à partir du plafond et du plancher qui y sont calculés.

Exercice 9f : Rassembler le tout en une fonction `enveloppe points`.

Exercice 10 : Prouver la terminaison du programme.

Exercice 11 : Déterminer sa complexité dans le pire des cas (indice : si c'est pire que le résultat trouvé dans la section précédente, c'est qu'il y a une erreur quelque part), en établissant la formule de récurrence associée.

Il existe de nombreux autres algorithmes pour calculer une enveloppe convexe, notamment dans le plan, en particulier Quickhull, qui est aussi un algorithme DPR.