TP 5 : Diviser pour régner
Application à la recherche de deux points les plus proches

I Introduction

Lors de cette séance, nous allons nous intéresser au problème suivant : étant donné un ensemble de points du plan, identifier le couple de points les plus proches au sens de la distance euclidienne. Ce type d'algorithme voit son utilité dans les transports aériens ou maritimes par exemple.

Nous représenterons un point en CAML par un **couple de flottants** (x, y), (type `float * float`) représentant ses coordonnées dans un repère orthonormé.

L'ensemble des points considérés sera quant à lui stocké dans un **tableau `t`** (type `float * float array`). Nous supposerons toujours que tous les couples stockés dans t sont distincts.

II Une première solution

Considérons deux points p et p' de coordonnées respectives (x, y) et (x', y'). Leur distance euclidienne est donnée par la formule $\|p - p'\| = \sqrt{(x - x')^2 + (y - y')^2}$.

1. Écrire une fonction `dist` qui calcule la distance euclidienne séparant deux points.

Une méthode « naïve » permettant de déterminer les deux points les plus proches dans un tableau t consiste à considérer successivement tous les couples de points possibles, ce qui peut se faire facilement à l'aide de deux boucles `for` .

2. Écrire une fonction `plus_proches` qui prend pour argument un tableau de points `t` et retourne un couple de points $((x, y), (x', y'))$ situés à une distance minimale l'un de l'autre.

Suggestion :

— Commencer par créer une référence d (initialisée par exemple avec `dist t.(0) t.(1)`) qui contiendra tout au long du déroulement de l'algorithme la plus petite distance trouvée ; ainsi que deux références `s_i` et `s_j` contenant les indices des points qui permettent d'atteindre cette distance.

— Utiliser deux boucles imbriquées pour considérer tous les couples de points communs.

— Mettre à jour les trois références à chaque fois qu'un couple (i, j) tel que $\|t.(i) - t.(j)\| < !d$ est trouvé.

— Attention il y a ne contrainte sur i et j que je n'ai pas explicitée et qui est très importante pour que l'algorithme fonctionne correctement...

Vous trouverez dans le fichier CAML qui accompagne cet énoncé une fonction qui génère un tableau de flottants dont les coordonnées sont comprises en 0 et 400.

Vous trouverez également une fonction de visualisation de l'ensemble de points. Un quadrillage avec des petits carrés 20×20 de côté. Attention pour fermer cette fenêtre **il ne faut cliquer sur sa croix rouge en haut à droite, mais appuyer sur une touche du clavier**. Si vous cliquez sur la croix rouge, vous avez fermer la représentation graphique de la fenêtre mais le processus qui l'a créée ! Il faut alors retourner dans emacs et choisir le menu **Tuareg**,

Interactive Mode puis **Interrupt OCaml REPL**, voire si nécessaire **Kill OCaml REPL** (et dans ce cas il faudra réévaluer toutes les phrases OCaml déjà écrites...

Utilisez ces deux fonctions pour vérifier que votre fonction rend bien ce qu'il faut...

3. Évaluer le coût d'un appel à la fonction `plus_proches` en fonction de la taille du tableau t.

On va maintenant s'intéresser à la mise au point d'un algorithme permettant de résoudre ce même problème en comlpexité temporelle $\mathcal{O}(n \log n)$. Cet algorithme nécessite de disposer de deux copies du tableau t, l'une triée par abscisses croissantes et l'autre par ordonnées croissantes.

Pour se focaliser sur la partie diviser pour régner vous utiliserez la fonction tri fusion présente dans le fichier Caml accompagnant l'énoncé. Attention, pour les besoins de ce TP cette fonction prend en argument, en plus du tableau à trier, une fonction booléenne `ordre` de type `'a * 'a -> bool` permettant de comparer deux éléments du tableau : `ordre a1 a2` doit rendre `true` si a_1 est strictement inférieur à a_2 et `false` sinon. En effet ici on travaille avec des tableaux de points et l'ordre proposé par Caml sur le type `float * float` est l'ordre lexicographique, ce qui ne nous convient pas.

Si vous avez le temps vous étudierez la dernière partie du TP le tri par sélection.

III Diviser pour régner

On revient donc au problème initial, à savoir déterminer parmi un ensemble de points un couple de points situés à une distance minimale.

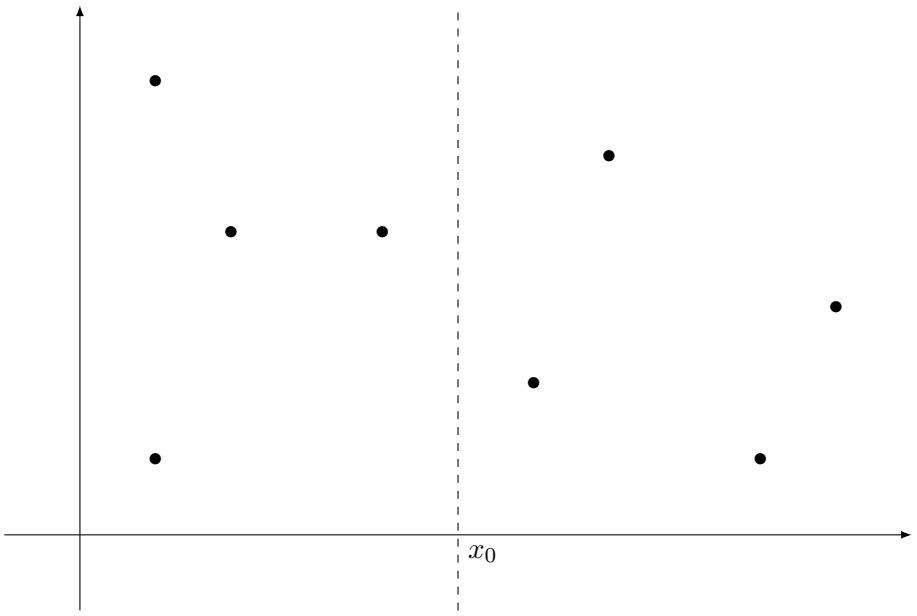
On suppose disposer de deux copies de l'ensemble des n points :

— un tableau t_x, trié par abscisses croissantes,

— un tableau t_y, trié par ordonnées croissantes.

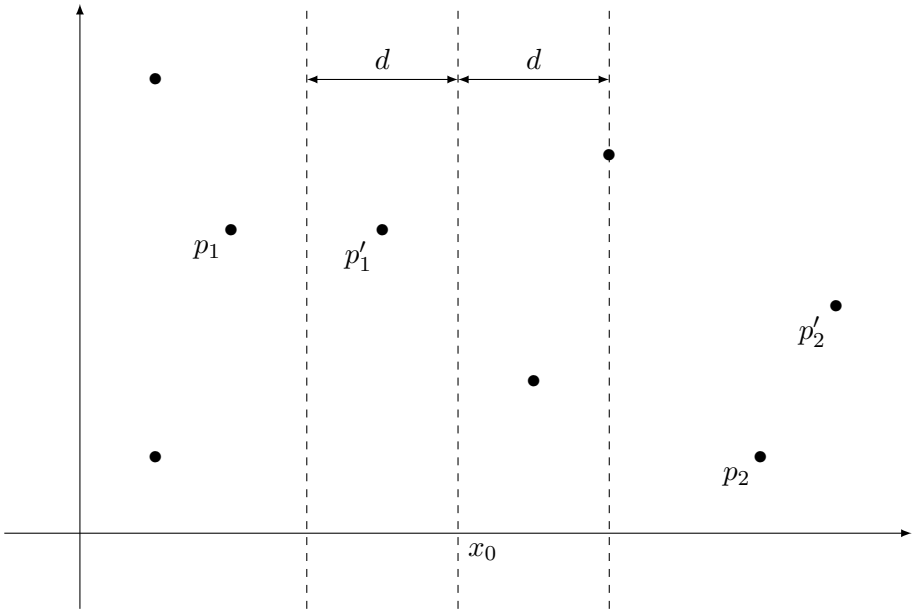
On pose $k = n/2$ (division entière) et on considère les deux sous-tableaux $t_1 = [|t.(0); \ldots; t.(k)|]$ et $t_2 = [|t.(k+1); \ldots; t.(n-1)|]$.

Soit x_0 un flottant plus grand que les abscisses des points de t_1 et plus petit que les abscisses des points de t_2. Le plan est divisé en deux parties :



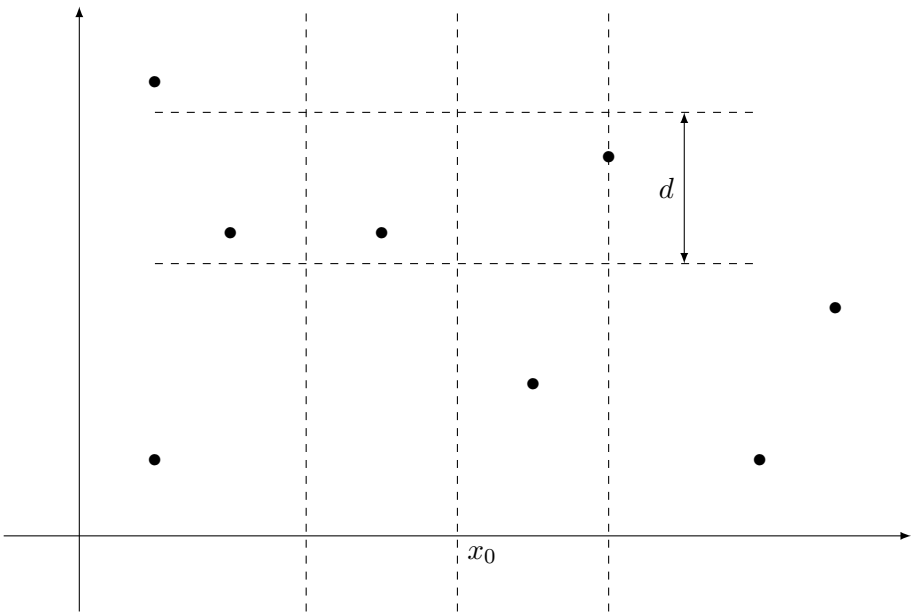
On note $p_1 = (x_1, y_1)$ et $p'_1 = (x'_1, y'_1)$ les deux points les plus proches dans le tableau t_1 (avec $x_1 \leqslant x'_1$) ainsi que $p_2 = (x_2, y_2)$ et $p'_2 = (x'_2, y'_2)$ deux points les plus proches dans t_2. On note enfin $d = \min(\|p_1 - p'_1\|, \|p_2 - p'_2\|)$.

Si deux points du tableau t sont à une distance strictement inférieure à d alors nécessairement l'un fait partie de t_1 et l'autre de t_2 et ils sont dans la bande d'abscisses $[x_0 - d; x_0 + d]$.



4. Écrire une fonction `seltz` qui prend pour argument un tableau t_y ainsi que deux flottants x_{min}, x_{max} et qui sélectionne dans ce tableau les points dont l'abscisse est comprise entre x_{min} et x_{max}. Le résultat sera rendu dans un tableau t_z. L'ordre initial de t_y devra être bien sûr conservé dans t_z.

   On note donc t_z le tableau des points situés dans la bande, obtenu grâce à un appel à la fonction précédente.
   On admet que dans chaque tranche de hauteur d de cette bande, il ne peut y avoir plus de huit points. On en déduit que si deux points de t_z sont à une distance inférieure ou égale à d alors ils sont situés à au plus sept cases l'un de l'autre dans le tableau t_z.



5. Écrire une fonction `plus_proches_centre` qui prend pour argument le tableau t_z et retourne le couple de points les plus proches, tels que ces deux points soient situés à

au plus 7 cases l'un de l'autre dans le tableau t_z. On pourra s'inspirer de la fonction `plus_proches` précédente.

On déduit de ce qui précède un algorithme récursif permettant de trouver les deux points les plus proches à partir des deux tableaux t_x et t_y :

- Découper le tableau t_x en deux tableaux $t_{x,1}$ et $t_{x,2}$ de longueur moitié, à condition que chaque tableau puisse contenir au moins deux points (dans le cas contraire on détermine le couple des deux points les plus proches par appel à la fonction `plus_proche` écrite au tout début du TP).
- Déterminer $p_1 = (x_1, y_1)$, $p'_1 = (x'_1, y'_1)$ (couple de points les plus proches de $t_{x,1}$), $p_2 = (x_2, y_2)$ et $p'_2 = (x'_2, y'_2)$ (couple de points les plus proches de $t_{x,2}$) ainsi que la distance d.
- Calculer t_z. S'il contient au moins deux points, appliquer la fonction `plus_proches_centre` à t_z pour obtenir $p = (x, y)$ et $p' = (x', y')$.
- Retourner le résultat, c'est à dire la paire de points à distance minimale parmi p_1, p'_1 ; p_2, p'_2 et p, p'.

6. Écrire les deux fonctions `ordre_x` et `ordre_y` à passer en paramètre à la fonction `tri_fusion` pour trier un tableau de points suivant les abscisses ou les ordonnées croissantes.
7. Écrire en appliquant ce procédé une nouvelle fonction `plus_proches_dpr` . Évaluer sa complexité en fonction du nombre de points.

On repasse maintenant à l'étude d'un tri.

IV Tri d'un tableau

Soit $t = [|x_0; x_1; \ldots; x_{n-1}|]$ un tableau. Trier t consiste à permuter le contenu des cases du tableau t de façon à obtenir le tableau $[|x_{\sigma(0)}; x_{\sigma(1)}; \ldots; x_{\sigma(n-1)}|]$ avec $x_{\sigma(0)} \leq x_{\sigma(1)} \leq \ldots \leq x_{\sigma(n-1)}$.

L'algorithme de *tri par sélection* adopte le principe suivant : pour i variant de 0 à $n-2$, chercher l'indice $k \geq i$ de la case de la partie du tableau située à droite de la case i contenant le plus petit élément ; puis permuter le contenu des cases k et i.

Par exemple, pour trier le tableau $[|4; 3; 1; 2|]$, on passe par les étapes suivantes :

$[\|\mathbf{4}; 3; \mathbf{1}; 2\|]$	$i = 0$	$k = 2$
$[\|1; \mathbf{3}; 4; \mathbf{2}\|]$	$i = 1$	$k = 3$
$[\|1; 2; \mathbf{4}; \mathbf{3}\|]$	$i = 2$	$k = 3$
$[\|1; 2; 3; 4\|]$		

8. Écrire une fonction `swap` qui prend pour argument un tableau t, deux entiers a et b et permute le contenu des cases d'indices a et b. La fonction ne créera pas un nouveau tableau, elle se contentera de modifier *en place* le tableau t. Elle sera donc de type 'a array -> int -> int -> unit.
9. Écrire une fonction `sel` qui prend pour argument un tableau t ; deux entiers a et b et retourne l'indice de la case du sous-tableau $t.(a) \ldots t.(b)$ qui contient le plus petit élément pour l'ordre $<$.
10. En déduire une fonction `tri` qui effectue le tri d'un tableau.

La fonction qui vient d'être écrite permet de trier un tableau pour la relation d'ordre $<$. Sur les couples, cet ordre est en Caml l'ordre lexicographique. On peut cependant imaginer ou avoir besoin d'autres relations d'ordre sur des couples.

11. Adapter les fonctions `sel` et `tri` de sorte qu'elle prennent également en argument une fonction implantant une relation d'ordre de type 'a * 'a -> bool . On nommera `sel'` et `tri'` ces deux nouvelles fonctions.