

# MPSI : TP Programmation dynamique

## Deux plus proches points du plan

### 0.1 Présentation

Considérons un nuage de  $n \geq 3$  points du plan définis par leurs coordonnées cartésiennes dans un repère orthonormé. La distance entre deux points  $P, Q$  est la norme euclidienne de  $\overrightarrow{PQ}$ . Le problème des \*deux plus proches points du plan\* consiste à identifier les deux points les plus proches dans le nuage.

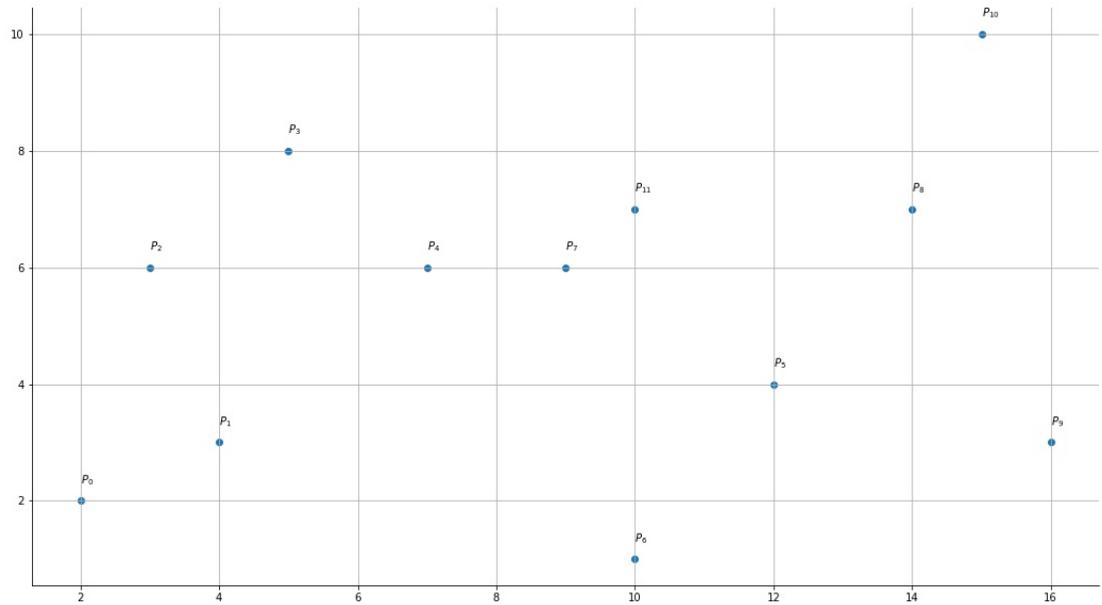


FIGURE 1 – Un nuage de points

Une solution serait de calculer les distances entre toute paire de sommets et de choisir la

meilleure. On aurait alors une complexité de l'ordre de  $\frac{n(n-1)}{2}$ . On peut cependant faire beaucoup mieux.

Notons  $a$  le tableau qui rassemble les coordonnées du nuage. On construit deux tableaux  $a_x$  et  $a_y$  qui contiennent respectivement les éléments de  $a$  triés par abscisses croissantes et par ordonnées croissantes.

Si  $n \leq 3$ , un algorithme direct détermine les deux points les plus proches parmi les deux ou trois points, ce qui constitue un cas de base pour la suite de l'algorithme. Si  $n \geq 4$ , l'algorithme procède comme suit :

**Diviser** Le tableau  $a_x$  est divisé en deux sous-tableaux  $a_{x,g}$  et  $a_{x,d}$  de tailles  $\lfloor n/2 \rfloor$  et  $\lceil n/2 \rceil$  respectivement. Les points stockés dans ces sous-tableaux sont situés de part et d'autre de la droite verticale  $\Delta$  d'équation  $x = x_{\text{med}}$  ou  $x_{\text{med}} = \lfloor n/2 \rfloor$ . Cette valeur  $x_{\text{med}}$  est l'abscisse du point  $a_x[\lfloor n/2 \rfloor - 1]$

**Régner** L'algorithme, appelé récursivement sur  $a_{x,g}$  et  $a_{x,d}$ , détermine les deux distances minimales  $d_g$  et  $d_d$  ainsi que les deux couples de points associés à ces distances. Seul le couple de points correspondant à  $d = \min(d_g, d_d)$  est conservé (voir figure 2).

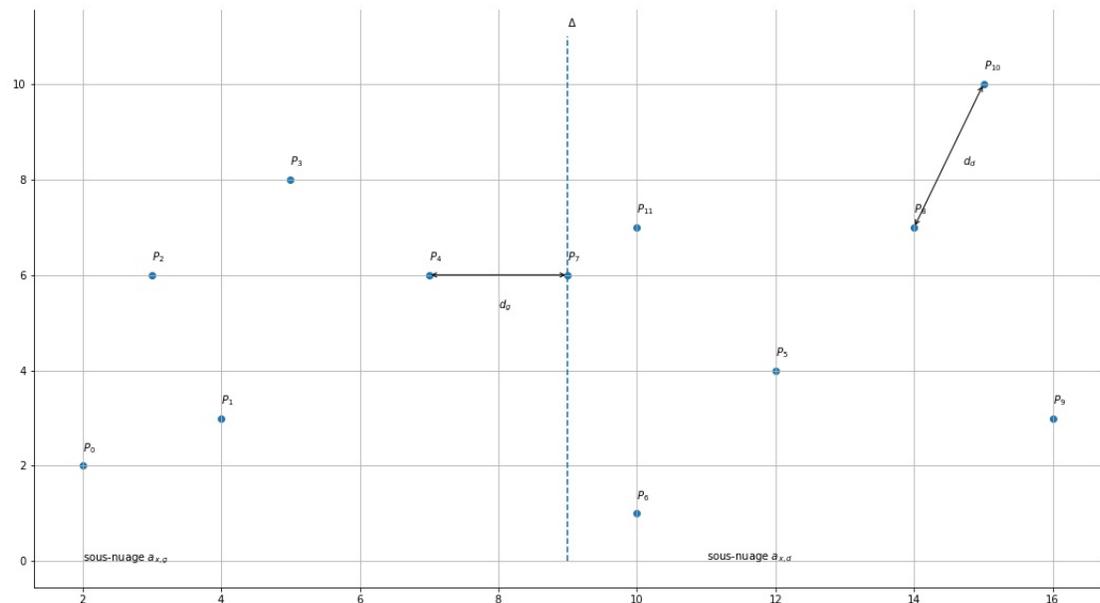


FIGURE 2 – Exploration des 2 sous-nuages de points. Celui de gauche contient aussi la droite verticale.

**Rassembler** Une fois un couple de plus proches points identifiés dans  $a_{x,g}$  ou  $a_{x,d}$ , on étudie une bande verticale de largeur  $2d$  centrée sur  $\Delta$ . Il est en effet possible que deux points de cette bande soient plus proches que ceux déjà trouvés. Il faut donc une étape de plus pour examiner les points de la bande. Cette recherche se fait en considérant le sous-tableau  $a_{\text{bande}}$  de  $a_y$  qui ne contient que les points dont les abscisses sont comprises entre  $x_{\text{med}} - d$  et  $x_{\text{med}} + d$ .

Sur la figure 3, on constate que deux points de la bande verticale ont une distance inférieure à celles trouvées dans les sous-nuages gauche et droits. En effet ces deux points appartenant à deux sous-nuages différents ne pouvaient pas être détectés.

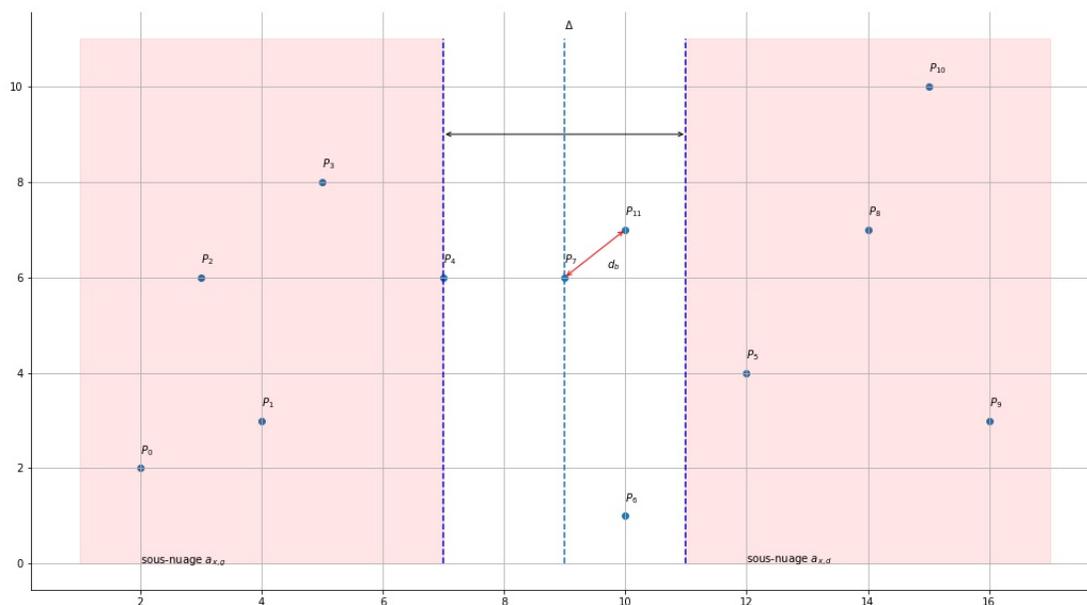


FIGURE 3 – Une bande verticale

**Rassembler (suite)** On montre que pour chaque point de la bande verticale, il suffit d'examiner uniquement les 7 points qui le suivent dans le tableau  $a_{\text{bande}}$ .

Pour cela, considérons un point  $P_i(x, y)$  de la bande verticale et cherchons les points  $P$  de la bande tels que  $P_iP < d$  et  $P$  est au-dessus de  $P_i$ . Ils sont nécessairement dans le rectangle  $R$  de hauteur  $d$  et de largeur  $2d$  centré sur  $\Delta$  dont  $P_i$  est sur le côté inférieur (voir figure 4).

Découpons notre rectangle en 8 carrés de de côtés  $\frac{d}{2}$  comme sur la figure 5. Chacun des petits carrés contient au plus un point du nuage initial (car la distance maximale entre deux points d'un carré est la diagonale -de longueur  $\sqrt{2}\frac{d}{2} = \frac{1}{\sqrt{2}}d$ - qui est strictement inférieure à  $d$ ). En

conclusion, il y a au plus 8 points de la bande verticale dans le rectangle  $R$ . Au total, il y a au maximum 7 points en plus de  $P_i$  dans le rectangle!

Il suffit donc de parcourir tous les points de la bande et d'examiner les 7 points suivants pour déterminer s'il existe une paire de points dont la distance est inférieure à  $d$ . Ce sera notre paire optimale si on en trouve une, sinon la paire optimale est dans un des deux sous-nuages  $a_g$  ou  $a_d$ .

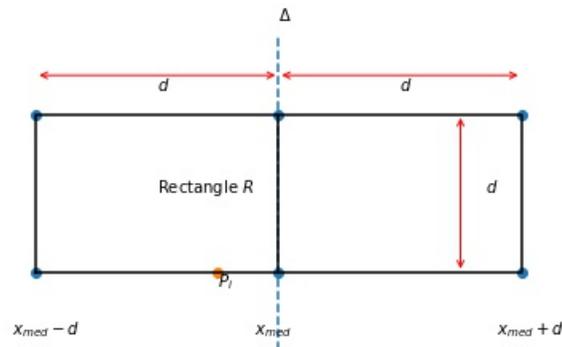


FIGURE 4 – Un rectangle inscrit dans la bande verticale.

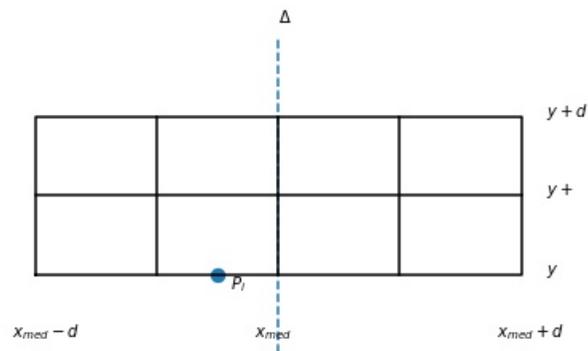


FIGURE 5 – Décomposition du rectangle  $R$  en 8 carrés de côté  $\frac{d}{2}$ .

## 0.2 Questions

On considère les types suivants :

```

1 | type point = float * float;;
2 | type cloud = point array;;

```

**Q.1** Écrire la fonction `dist p1 p2` qui calcule la distance euclidienne entre deux points du plan donnés par leurs tuples de coordonnées.

```

1 | # dist (1.,2.) (-2.,3.);;
2 | - : float = 3.16227766016837952

```

**Q.2** Écrire la fonction `sort_abs a` qui retourne une version triée par abscisse croissante d'un tableau de points. Le tableau n'est pas modifié. Faire de même avec `sort_ord a` qui trie selon les ordonnées croissantes.

```

1 | # let cloud =
2 |   [(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.); (12., 4.); (10., 1.);
3 |     (9., 6.); (14., 7.); (16., 3.); (15., 10.); (10., 7.)] in
4 | abs_sort cloud;;
5 | - : point array =
6 | [(2., 2.); (3., 6.); (4., 3.); (5., 8.); (7., 6.); (9., 6.); (10., 7.);
7 |   (10., 1.); (12., 4.); (14., 7.); (15., 10.); (16., 3.)]
8 | # let cloud =
9 |   [(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.); (12., 4.); (10., 1.);
10 |    (9., 6.); (14., 7.); (16., 3.); (15., 10.); (10., 7.)] in
11 | ord_sort cloud;;
12 | - : point array =
13 | [(10., 1.); (2., 2.); (16., 3.); (4., 3.); (12., 4.); (9., 6.); (3., 6.); (7.,
14 |   6.); (10., 7.); (14., 7.); (5., 8.); (15., 10.)]
15 | #

```

**Q.3** Écrire une fonction `make_pos a` qui prend en paramètre un nuage de points et retourne le dictionnaire (point, position dans le nuage).

```

1 | # let cloud =
2 |   [(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.); (12., 4.); (10., 1.);
3 |     (9., 6.); (14., 7.); (16., 3.); (15., 10.); (10., 7.)] in
4 | let h = make_pos cloud in
5 | Hashtbl.find h (4., 3.), Hashtbl.find h (10., 7.);;
6 | - : int * int = (1, 11)

```

**Q.4** Écrire une fonction `make_new_ay ay d lo mid hi` qui prend en paramètres un tableau (censé être ordonné par ordonnées croissantes), un dictionnaire `d` des positions des points dans un autre tableau (en fait, ce sont les positions dans  $a_x$ ), une borne inférieure `lo` des positions étudiées, une borne médiane `mid` et une borne supérieure `hi` exclue.

La fonction retourne deux sous-tableaux de  $a_y$  :

- le sous-tableau des points de  $a_y$  qui occupent des positions entre `lo` et `mid-1` dans  $a_x$ .
- le sous-tableau des points de  $a_y$  qui occupent des positions entre `mid` et `hi-1` dans  $a_x$ .

```

1 | # let cloud =
2 |   [(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.); (12., 4.); (10., 1.);
3 |     (9., 6.); (14., 7.); (16., 3.); (15., 10.); (10., 7.)] in

```

```

3 | let ax = abs_sort cloud and ay = ord_sort cloud in
4 | let h = make_pos ax in
5 | ax, ay, make_new_ay ay h 3 5 8;;
6 | - : point array * point array *
7 | ((float * float) array * (float * float) array)
8 | =
9 | ([|(2., 2.); (3., 6.); (4., 3.); (5., 8.); (7., 6.); (9., 6.); (10., 7.);
10 | (10., 1.); (12., 4.); (14., 7.); (15., 10.); (16., 3.)|],
11 | [||(10., 1.); (2., 2.); (16., 3.); (4., 3.); (12., 4.); (9., 6.); (3., 6.);
12 | (7., 6.); (10., 7.); (14., 7.); (5., 8.); (15., 10.)|],
13 | ([|(7., 6.); (5., 8.)|], [||(10., 1.); (9., 6.); (10., 7.)|]))

```

**Q.5** Écrire la fonction `closest_pair_small a` qui prend en paramètre un tableau d'au plus 3 points et retourne le couple de points les plus proches. Une exception est soulevée si le tableau est trop long. Cette fonction auxiliaire gère donc les cas de base de notre algorithme.

```

1 | a = [(5,2),(4,6),(2,3)]
2 | closest_pair_small(a)

```

**Q.6** Écrire la fonction `make_strip a xmin xmax` qui construit la bande verticale à partir d'un tableau de points  $a$  (en pratique rangés par ordre croissant d'ordonnées), d'une abscisse gauche  $x_{\min}$  et d'une abscisse droite  $x_{\max}$ .

```

1 | # let cloud =
2 | [||(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.); (12., 4.); (10., 1.);
3 | (9., 6.); (14., 7.); (16., 3.); (15., 10.); (10., 7.)|] in
4 | let xmin = 1.5 and xmax = 7.5 in
5 | make_strip cloud xmin xmax;;
6 | - : (float * float) array =
7 | [||(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.)|]

```

**Q.7** Écrire la fonction `closest_pair_in_strip a` qui prend en paramètres une bande verticale comme calculée par la fonction précédente. La fonction retourne les deux points les plus proches dans le tableau  $a$  en examinant à chaque tour les 7 points qui suivent le point courant.

```

1 | # let cloud =
2 | [||(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.); (12., 4.); (10., 1.);
3 | (9., 6.); (14., 7.); (16., 3.); (15., 10.); (10., 7.)|] in let ay =
4 | ord_sort cloud in
5 | closest_pair_in_strip ay;;
6 | - : point * point = ((9., 6.), (10., 7.))

```

**Q.8** Écrire la fonction `closest_pair : cloud -> point * point` qui calcule les deux points les plus proches du nuage  $a$  en appliquant l'algorithme.

```

1 | # let cloud =
2 | [||(2., 2.); (4., 3.); (3., 6.); (5., 8.); (7., 6.); (12., 4.); (10., 1.);
3 | (9., 6.); (14., 7.); (16., 3.); (15., 10.); (10., 7.)|] in
4 | closest_pair cloud;;
5 | - : point * point = ((9., 6.), (10., 7.))

```

**Q.9** Donner une relation de récurrence suivie par la complexité temporelle au pire  $C(n)$  de l'appel `closest_pair a` si  $|a| = n$ . En déduire cette complexité.