PCSI 1 **Devoir surveillé d'informatique n° 1 :** 2023/2024

**Jeudi 29 février**

**1h30**

**Les fonctions doivent être commentées.**

**L'indentation doit être représentée par un trait vertical.**

# Points proches dans le plan

Ce problème, pouvant par exemple survenir dans le domaine de la navigation maritime, vise à déterminer, dans un nuage de points du plan, la paire de points les plus proches. Il est constitué de trois parties dépendantes.

Formellement, on suppose qu'on dispose de $n$ points dans le plan $(M_0, M_1, \dots, M_{n-1})$ dans un ordre quelconque pour le moment. Ils seront représentés en Python par deux listes de flottants de taille $n$ : `coords_x` et `coords_y`, donnant respectivement les abscisses et les ordonnées des points. On dira ainsi que $M_i$ est le point d'indice $i$, qu'il a pour abscisse $x_i$ = `coords_x [i]` et pour ordonnée $y_i$ = `coords_y [i]`. On supposera que `coords_x` et `coords_y` sont des variables globales, qu'on ne modifiera jamais au cours de l'exécution de l'algorithme.

## I Approche exhaustive

On utilise la distance euclidienne définie par $d(M_i, M_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$.

1. Écrire une fonction `distance(i, j)` qui renvoie la distance entre les points $M_i$ et $M_j$. On utilisera la fonction `sqrt` après l'avoir importée.
2. Écrire une fonction `plus_proche()` qui renvoie, à l'aide d'une recherche exhaustive, le couple d'entiers des indices `i` et `j` des deux points les plus proches du nuage de points.

## II Quelques outils pour s'améliorer

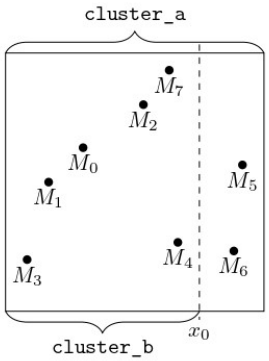
On souhaite maintenant obtenir la distance entre les deux points les plus proches avec une meilleure efficacité. Cette partie introduit des fonctions utiles pour la mise en œuvre de cet algorithme.

Dans toute la suite, on considère que $l$ est une liste de coordonnées de points.
On considèrera l'exemple suivant : $l_{ex}$=[(1,2),(5,3),(2,4),(1,1),(3,2)]

3. (a) Écrire une fonction `tri_liste_abscisses(l)` qui renvoie la liste `l` triée suivant l'ordre des abscisses croissantes. Pour cela, on utilisera l'un des principes de tri vus en cours et on nommera la méthode utilisée.
   Par exemple `tri_liste_abscisses(`$l_{ex}$`)` renvoie : `[(1, 2), (1, 1), (2, 4), (3, 2), (5, 3)]`
   (b) Écrire une fonction `ListeDict(l)` qui renvoie un dictionnaire dont les clés sont les coordonnées des points de `l` et dont les valeurs sont les indices des points.
   Par exemple `ListeDict(`$l_{ex}$`)` renvoie : `{(1, 2) : 0, (5, 3) : 1, (2, 4) : 2, (1, 1) : 3, (3, 2) : 4}`
   (c) Écrire une fonction `tri_indice_abscisses(l)` qui renvoie la liste des indices des points de `l` triés suivant l'ordre des abscisses croissantes.
   Par exemple `tri_indice_abscisses(`$l_{ex}$`)` renvoie : `[0, 3, 2, 4, 1]`
   (d) On suppose qu'on a écrit une fonction `tri_indice_ordonnees(l)` qui renvoie la liste des indices des points de `l` triés suivant l'ordre des ordonnées croissantes.
   Que renvoie `tri_indice_ordonnees(`$l_{ex}$`)` ?

On admettra que l'on dispose de deux listes de $n$ entiers `liste_x=tri_indice_abscisses(l)` (resp. `liste_y=tri_indice_ordonnees(l)`) contenant les indices des points du nuage triés par abscisses croissantes (resp. par ordonnées croissantes). On supposera désormais que deux points quelconques ont des abscisses et des ordonnées distinctes.

Dans toute la suite, un sous-ensemble de points sera décrit par un cluster. Un cluster est une matrice de deux lignes contenant chacune les mêmes numéros correspondant aux numéros des points dans le sous-ensemble considéré. Dans la première ligne, les points sont triés par abscisses croissantes ; dans la seconde, ils sont triés par ordonnées croissantes. La figure 1 donne la représentation de deux clusters.




```
liste_x = [3, 1, 0, 2, 7, 4, 6, 5]
liste_y = [3, 6, 4, 1, 5, 0, 2, 7]

cluster_a = [liste_x, liste_y]

cluster_b = [ [3, 1, 0, 2, 7, 4],
              [3, 4, 1, 0, 2, 7] ]
```

Figure 1 - Représentation en Python de deux clusters

Pour être efficace, notre algorithme ne doit pas re-trier les listes des indices de points à chaque étape. Nous allons donc définir une fonction qui permet d'extraire des indices d'un cluster et former ainsi un nouveau cluster plus petit.

4. Écrire une fonction `sous_cluster(cl, x_min, x_max)` qui prend en arguments un cluster `cl` et deux flottants `x_min` et `x_max`, et renvoie le sous-cluster des points dont l'abscisse est comprise entre `x_min` et `x_max` (au sens large).
5. Écrire une fonction `mediane(cl)` qui prend en entrée un cluster `cl` contenant au moins 2 points et renvoie une abscisse médiane, c'est-à-dire que la moitié (au moins) des points a une abscisse inférieure ou égale à cette valeur, et la moitié (au moins) des points a une abscisse supérieure ou égale à cette valeur.

# III Méthode sophistiquée

Le fonctionnement de l'algorithme est illustré par la figure 2 :

i Si le cluster contient deux ou trois points, on calcule la distance minimale en calculant toutes les distances possibles.

ii Sinon, on sépare le cluster en deux parties $G$ et $D$ qu'on supposera de tailles égales (éventuellement à un point près) suivant la médiane des abscisses, qu'on notera $x_0$.

iii Les deux points les plus proches sont soit tous les deux dans $G$, soit tous les deux $D$, soit un dans $G$ et un dans $D$.

iv On calcule récursivement le couple le plus proche dans $G$ et le couple le plus proche dans $D$. On note $d_0$ la plus petite des deux distances obtenues.

v On cherche s'il existe une paire de points $(M_1, M_2)$ telle que $M_1$ est dans $G$, $M_2$ dans $D$, et $d(M_1, M_2) < d_0$.

vi Si on en trouve une (ou plusieurs), on renvoie la plus petite de ces distances. Sinon, on renvoie $d_0$.

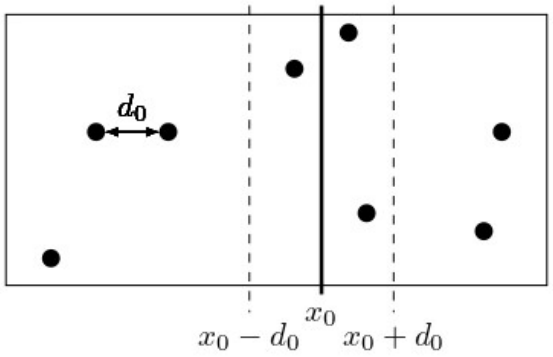



Figure 2 - Illustration de la méthode sophistiquée

6. Écrire une fonction `gauche(cl)` qui prend en argument un cluster `cl` contenant au moins deux points et renvoie le cluster constitué uniquement de la moitié (éventuellement arrondie à l'entier supérieur) des points les plus à gauche du cluster `cl`.

   On suppose qu'on dispose d'une fonction `droite(cl)` qui renvoie le cluster contenant tous les autres points du cluster `cl` n'appartenant pas au cluster renvoyé par la fonction `gauche(cl)`.
7. Justifier que l'on peut se contenter de chercher les points $M_1$ et $M_2$ de l'étape v de l'algorithme dans l'ensemble des points dont l'abscisse appartient à $I_0 = [x_0 - d_0, x_0 + d_0]$.
8. Écrire une fonction `bande_centrale(cl, d0)` qui prend en argument un cluster `cl` et un réel `d0`, et renvoie le cluster des points dont l'abscisse est dans $I_0 = [x_0 - d_0, x_0 + d_0]$.
9. En déduire une fonction `fusion(cl, d0)` qui prend en entrée un cluster de points dont toutes les abscisses sont dans un intervalle $[x_0 - d_0, x_0 + d_0]$, et renvoie la distance minimale entre deux points du cluster si elle est inférieure à $d_0$, ou $d_0$ sinon.
10. Écrire une fonction récursive `distance_minimale(cl)` qui prend en argument un cluster et utilise l'algorithme décrit plus haut pour renvoyer la distance minimale entre deux points du cluster.