Thème 6

Devoir maison - Diviser pour régner

NSI T^{ale}

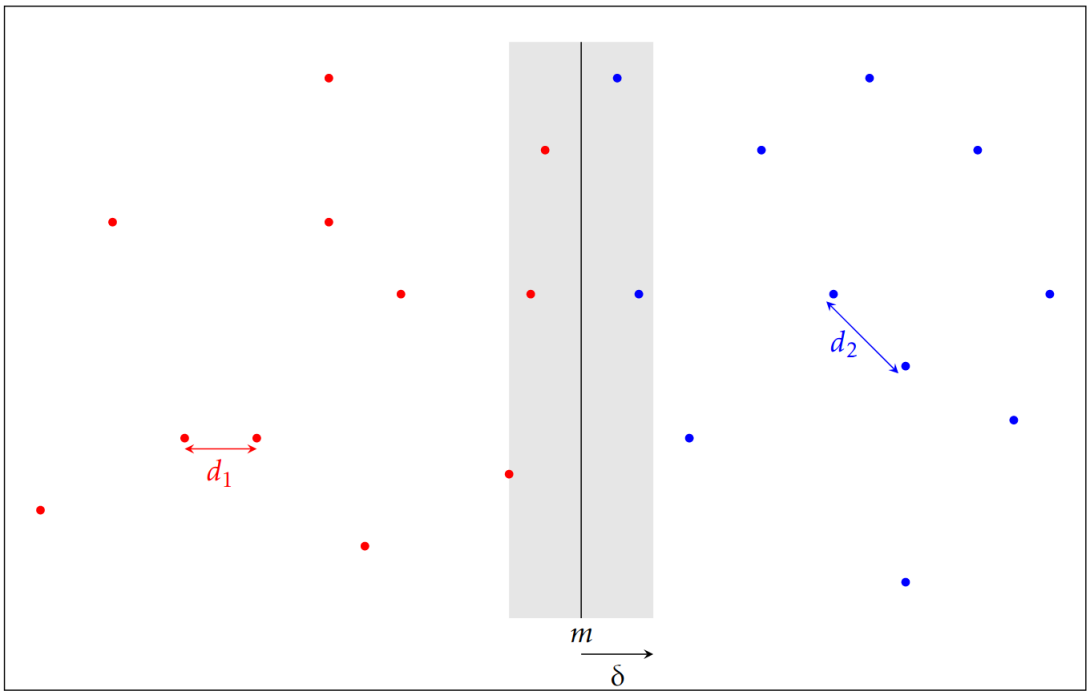
Distance minimale entre deux points dans un nuage

Nous nous intéressons au problème de la recherche des deux points les plus proches dans un nuage $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)$ de points distincts du plan.

L'algorithme naïf consiste à calculer la distance entre toutes les paires possibles et d'en extraire le minimum. Sachant qu'il y en a $\binom{n}{2}$, ceci conduit à un algorithme de coût quadratique $\mathcal{O}(n^2)$. Peut-on faire mieux en adoptant une stratégie diviser pour régner ? Oui !

Avant toute chose, nous allons avoir besoin d'ordonner ces points par abscisses croissantes, mais aussi par ordonnées croissantes. Nous allons donc déterminer deux tableaux `Tx` et `Ty`, le premier contenant les points classés par ordre croissant des abscisses (puis des ordonnées), et le second par ordre croissant des ordonnées (puis des abscisses). Nous savons que les tris peuvent être exécutés en $\mathcal{O}(n \log n)$.

Nous allons désormais supposer $\mathtt{T} = \{p_1, p_2, \ldots, p_n\}$ avec $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n$.



Séparons alors les points de T en deux parties sensiblement égales à gauche et à droite de la ligne médiane : $\mathtt{T}_G = \{p_1, p_2, \ldots, p_k\}$ et $\mathtt{T}_D = \{p_{k+1}, \ldots, p_{n-1}, p_n\}$, avec $k = n//2$, et notons δ_G et δ_D les distances minimales entre respectivement deux points de $\mathtt{T}_G$ et deux points de $\mathtt{T}_D$. Trois cas sont possibles pour la distance minimale δ entre deux points de T :

- elle est atteinte entre deux points de $\mathtt{T}_G$, auquel cas $\delta = \delta_G$;
- elle est atteinte entre deux points de $\mathtt{T}_F$, auquel cas $\delta = \delta_D$;
- elle est atteinte entre un point de $\mathtt{T}_G$ et un point de $\mathtt{T}_D$.

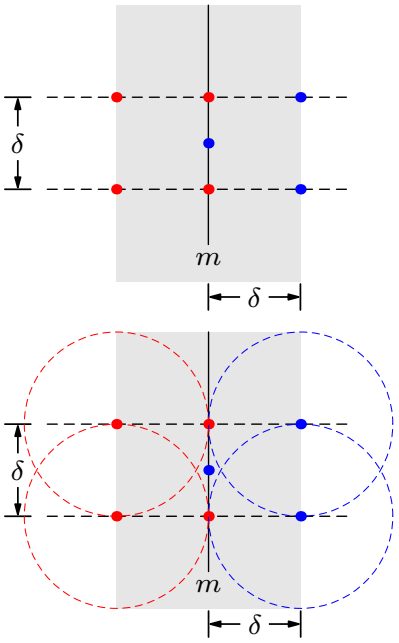
Le calcul de $\delta = min(d_1, d_2)$ permet d'éliminer l'un ou l'autre des deux premiers cas. Et dans le troisième cas, les deux points que nous recherchons se trouvent dans la bande verticale délimitée par les abscisses $med - \delta$ et $med + \delta$ avec $med = \dfrac{x_k + x_{k+1}}{2}$

Nous pouvons, en coût linéaire, déterminer les points `B` $= (p'_i,\ldots,p'_j)$ de cette bande verticale en les ordonnant par ordonnée croissante à l'aide du tableau `Ty` (il suffit de parcourir une fois `Ty` en ne gardant que les points dont les abscisses sont comprises entre $m-\delta$ et $m+\delta$).

Pour conclure, il reste à observer (c'est le point crucial) que dans chaque tranche de hauteur δ de cette bande ne peuvent se trouver qu'au plus sept points : quatre d'un côté et trois de l'autre.

Autrement dit : la différence entre les ordonnées des éléments d'indice $i+7$ et d'indice i est supérieure à δ.

Il s'en suit un coût linéaire pour la recherche dans la bande de largeur δ.



En définitive, le coût de cet algorithme (sans compter le pré-traitement des tableaux `Tx` et `Ty`) vérifie une relation de récurrence de la forme : $\mathcal{C}_n = 2\times\mathcal{C}_{n//2}+\mathcal{O}(n)$. Un théorème mathématique appelé le *Master Theorem* prouve alors que $\mathcal{C}_n = \mathcal{O}(n\log n)$.

Travail préliminaire

Exercice 1 Écrire une fonction `distance(p, q)` qui renvoie la distance entre deux points.

Exercice 2 Écrire une fonction `tableau(n)` qui renvoie un tableau `T` de n points aléatoirement choisis, définis par leurs coordonnées dans un repère du plan.

> **Attention :**
> Il ne doit pas y avoir de **doublon** dans le tableau de points : on obtiendrait une distance minimale égale à zéro, voire même une exception *index out of range*.

Exercice 3 Écrire une fonction `sous_tableaux_tries(t)` qui prend en entrée un tableau `T` et renvoie deux tableaux `tx` et `ty` comme définis précédemment : le premier contenant les points classés par abscisses croissantes (puis par ordonnées croissantes), le second les points classés par ordonnées croissantes (puis par abscisses croissantes). Ce travail -fait en double puisqu'il y a les mêmes points !- permettra un coût en $\mathcal{O}(n\log(n))$.
On utilisera les fonctions ci-dessous comme clefs de tri (attribut `key`) de la fonction **`sorted`**.

```
def absc(p):
    return p[0],p[1]      #tri par l'abscisse puis par l'ordonnée
def ordo(p):
    return p[1],p[0]      #tri par l'ordonnée puis par l'abscisse
```

Exercice 4 Ecrire une fonction `plus_courte_distance_naive(t)` qui prend en entrée un tableau `T` (**`list`**) de points, définis par des tuples (abscisse, ordonnée), et renvoie en sortie le triplet `(p, q, delta)` avec les deux points les plus proches (p et q) ainsi que la distance entre eux (delta). On utilisera un algorithme naïf de coût $\mathcal{O}(n^2)$.

La fonction récursive : `plus_courte_dist_dpr`

On veut écrire une fonction "plus courte distance diviser-pour-regner" : `plus_courte_dist_dpr` qui prend en argument `tx` et `ty` , et met en œuvre la méthode proposée ci-dessus.
Les tâches à accomplir sont les suivantes :

Étape	**Fonction appelée**
Une première fois, on scinde le tableau `t` en deux sous-ensembles de taille égale, à 1 près : `tx` et `ty`	`sous_tableaux_tries(t)`
• Descente récursive	`def plus_courte_dist_dpr(tx,ty)` Test sur la taille de `t`
↪ En dessous du seuil de 4 points on appelle la méthode non-récursive de calcul de distance. On obtient δ_G et δ_D les distances minimales à gauche et à droite de la ligne médiane.	`plus_courte_distance_naive(t)`

↻ Pour un nombre de points supérieur ou égal à 4, on scinde chaque sous-ensemble en deux parties de taille égale, à 1 près puis on fait l'appel récursif à la fonction sur les tableaux `tgx` et `tdx` d'une part, `tgy` et `tdy` d'autre part.	`scinde_tableaux(tx,ty)`
	Appel récursif :
	`plus_courte_dist_dpr(tgx,tgy)`
	`plus_courte_dist_dpr(tdx,tdy)`

• Combinaison	Test et sélection entre δ_G et δ_D.
On crée une bande de largeur 2δ, où $\delta = min(\delta_G , \delta_D)$.	`bandeVerticale(ty,delta,medx)`
Dans cette bande, pour chaque point, on calcule les distances avec les 7 points qui suivent ce point dans le tableau `ty_bande`. Le minimum des distances calculées est δ_B. On garde la plus petite des distances δ_G, δ_D ou δ_B.	`plus_courte_dist_bande(ty_bande ,delta)` Test et sélection.
On renvoie le résultat : les deux points et la distance minimale.	Fin de la fonction récursive

Exercice 5 Ecrire une fonction `scinde_tableaux(tx,ty)` définie ainsi :
Entrées : un tableau tx trié dans l'ordre des abscisses puis ordonnées croissantes et un tableau ty trié dans l'ordre des ordonnées puis abscisses croissantes.
Sorties : l'abcisse du point médian, les quatre tableaux parties gauche et droite de tx et idem pour ty.

```
def scinde_tableaux(tx,ty):
    """scinde chaque tableau en deux tableaux et renvoie les sous-tableaux
   gauche et droite autour du point médian"""
    n = len(tx)
    tgx = tx[0 : n//2]
    tdx = tx[n//2:]
    tgy = [0]*(n//2)
    tdy = [0]*(n-n//2)
    ...
    return  med[0], tgx, tdx, tgy, tdy
```

En utilisant la documentation des fonctions ci-dessous, compléter leur implémentation:

Exercice 6

```
def bandeVerticale(ty, delta, medx):
    """Crée un tableau contenant les points issus du tableau ty et à une
   distance inférieure à delta autour de medx.
    Entrées : le tableau ty des points de la bande trié par ordonnée puis
   par abscisse, la diamètre delta, l'abscisse du point médian medx.
    Sortie : le tableau des points d'abscisses comprise entre medx-delta et
   medx+delta (et trié par ordonnée)"""
    ty_bande = [] #Tableau des points à déterminer
    ...
    return ty_bande
```

Exercice 7

```
def dist_min_bande(ty_bande, delta):
    """Entrées : le tableau ty_bande de points de la bande centrale trié par
    ordonnée puis abscisse ; delta la plus petite distance entre deux points
    situés à gauche ou deux points situés à droite.
    Sorties : Un quadruplet res = (b,p1,p2,delta) où b est un booléen qui
   vaut False s'il n'y a pas de point dans ty_bande à distance inférieure à
   delta et sinon p1, p2 sont les points les plus proches et delta leur
   distance."""
    n = len(ty_bande)
    min_d = delta    #minimum en dehors de la bande centrale
    res = (False, (0,0),(0,0),0)
    for i in range(n):
        ...
    return res
```

Exercice 8 En utilisant le tableau Étape/Fonction, compléter la fonction ci-dessous.

```
def plus_courte_dist_dpr(tx,ty):
    if len(tx) < 4:      #Seuil de la récursivité : appel de la fonction non-récursive
        ...
    else:        #Mise en place de la récursivité ("multiple" puisqu'on agit sur tx et ty)
        ...
        if delta1 < delta2:      #Combinaison
            ...
        else:
            ...
        ty_bande = bandeVerticale(ty, delta, medx)
        ...
        if b == True:        #Cas où le minimum est dans la bande
            p0, p1, delta = p0_bande, p1_bande, delta_bande
        return p0, p1, delta
```

Exercice 9 Compléter votre code avec :

1) Les tests sur les préconditions (tx et ty sont-ils vraiment triés? Y a-t-il des doublons?...)
2) Une mesure (ex:1000 points) des performances comparées de l'algorithme diviser-pour-regner (`plus_courte_dist_dpr`) et de l'algorithme naïf (`plus_courte_distance_naive`).