Examen final

Durée : 1 heure 30.

Avant de commencer

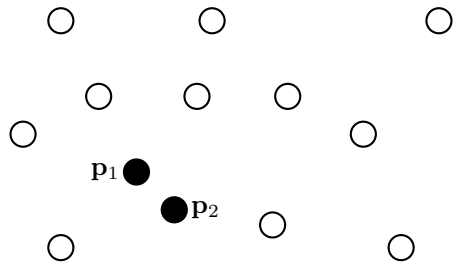
Le sujet est probablement trop long. Ne paniquez pas, le barême sera adapté en fonction de ce que le groupe aura réussi à faire. Ne vous concentrez pas sur les questions une par une, mais essayez de lire tout le sujet avant, car vous pourrez obtenir des indices pour certaines questions en lisant la suite. Un certain nombre de questions peuvent être répondues indépendamment des autres, donc ne restez pas coincés, essayez d'avancer rapidement sur ce que vous savez faire. Bon courage à tous.

1 Problème

Dans cet examen, nous allons étudier le problème de la recherche de la paire de points les plus proches dans un ensemble de points en 2D.

Données en entrée : en entrée, vous disposez d'un ensemble de n points 2D, rangés dans un tableau, sans ordre particulier. Un point $\mathbf{p}$ en 2D possède donc deux coordonnées, $\mathbf{p}.x$ et $\mathbf{p}.y$.

Sortie : nous cherchons à renvoyer en sortie la paire de points $\mathbf{p}_1$ et $\mathbf{p}_2$ les plus proches. Autrement dit, quels que soient les deux points choisis parmi les n points en entrée, la distance entre les deux points sera plus grande (ou égale) à la distance entre $\mathbf{p}_1$ et $\mathbf{p}_2$.



2 Algorithme naïf

Question 1 – Écrivez un algorithme naïf donnant le résultat, sans chercher à optimiser la complexité. Votre algorithme prendra donc en entrée le tableau de points et sa taille, et retournera la paire de points les plus proches. Dans votre algorithme, vous pouvez calculer la distance entre deux points $\mathbf{p}$ et $\mathbf{p}'$ en utilisant $d(\mathbf{p}, \mathbf{p}')$.

Question 2 – Quelle est la complexité de votre algorithme en fonction du nombre n de points en entrée ? Fournissez ici un ordre de grandeur et non un décompte précis du nombre d'itérations de vos boucles.

3 Diviser le problème

Pour améliorer la complexité de cet algorithme, nous allons envisager une approche du type diviser pour régner. Commençons donc par regarder comment diviser nos données.

Question 3 – Étant donné une valeur x_{mid}, écrivez un algorithme qui réordonne le tableau de points en deux portions, la portion de gauche telle que tous les points $\mathbf{p}$ soient tels que $\mathbf{p}.x \leq x_{mid}$ et la partie de droite telle que $\mathbf{p}.x > x_{mid}$. Votre algorithme prendra en paramètre le tableau de points, les indices de début (inclus) et fin (exclus) de la zone à partitionner, ainsi que x_{mid}. Il renverra l'indice du premier élément $\mathbf{p}$ de la plage [début, fin[tel que $\mathbf{p}.x > x_{mid}$ (ou fin si aucun point de la plage ne correspond). Par exemple sur le tableau tab suivant

0	1	2	3	4	5	6	7	8	9
(2,8)	(7,2)	(4,0)	(1,1)	(9,7)	(0,3)	(6,5)	(8,4)	(5,6)	(3,9)

l'appel à Partition(tab, 2, 6, 4) renverra 5 et donnera un tableau où les points d'indices entre 2 et 4 (inclus) ont un $x \leq x_{mid}$ et l'élément d'indice 5 un $x > x_{mid}$. Les autres éléments ne sont pas modifiés.

0	1	2	3	4	5	6	7	8	9
(2,8)	(7,2)	(4,0)	(1,1)	(0,3)	(9,7)	(6,5)	(8,4)	(5,6)	(3,9)

Question 4 – Quelle est la complexité de cet algorithme en fonction du nombre n de points ?

4 Un premier diviser pour régner

Pour commencer à aborder le problème sous l'angle diviser pour régner, nous vous proposons un premier algorithme (faux) pour essayer de résoudre le problème.

Fonction PlusProches (tab, début, fin) → (point, point)
  si fin- début < 2 **alors**
    Erreur : trop peu de points
  si fin- début ≤ 3 **alors**
    tester toutes les paires de points possibles pour les points de tab entre début (inclus) et fin (exclus)
    retourner la paire de points les plus proches
  sinon
    sélectionner x_{mid}
    milieu ← Partition(tab, début, fin, x_{mid})
    $(\mathbf{p}_1, \mathbf{p}_2)$ ← PlusProches(tab, début, milieu)
    $(\mathbf{q}_1, \mathbf{q}_2)$ ← PlusProches(tab, milieu, fin)

    si $d(\mathbf{p}_1, \mathbf{p}_2) < d(\mathbf{q}_1, \mathbf{q}_2)$ **alors**
      retourner $(\mathbf{p}_1, \mathbf{p}_2)$
    sinon
      retourner $(\mathbf{q}_1, \mathbf{q}_2)$

Question 5 – Dans les cas d'arrêt, quand nous testons toutes les paires de points possibles, combien y en a-t-il au maximum ?

Question 6 – Si la sélection de x_{mid} est réalisée en temps constant, le partitionnement en temps linéaire, et x_{mid} permet systématiquement de partitionner le tableau en deux parties égales, quelle est la complexité de cet algorithme ? On rappelle ci-dessous l'énoncé du Master Theorem :

Soit T une fonction de la forme

$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$

Le Master Theorem affirme que :

1. si $f(n) = \mathrm{O}(n^c)$ avec $c < \log_b a$, alors la complexité de f est trop faible devant la complexité liée à la récursion. Dans ce cas
$$T(n) = \Theta(n^{\log_b a}).$$
2. si $f(n) = \Omega(n^c)$ avec $c > \log_b a$, alors la complexité de f est prépondérante devant la complexité liée à la récursion. Dans ce cas il n'est pas toujours possible de déterminer la complexité finale à cause des appels récursifs à f. Si f respecte le critère de régularité : $\exists k < 1, \exists n_0 \geq 0, \forall n \geq n_0, af(n/b) \leq kf(n)$, alors
$$T(n) = \Theta(f(n)).$$

3. si $f(n) = \Theta(n^{\log_b a})$, alors les deux éléments interagissent et la complexité finale est donnée par

$$T(n) = \Theta(n^{log_b a} \log n).$$

Question 7 – Cet algorithme ne fournit pas le bon résultat. Illustrez un cas simple où le résultat de cet algorithme est mauvais.

Question 8 – Cet algorithme termine-t-il toujours ?

Question 9 – Imaginons désormais que le tableau tab est fourni trié selon x croissant. Le choix de x_{mid} et le partitionnement deviennent ainsi beaucoup plus simples. Quelle est alors la complexité de l'algorithme ?

5 Gérer le centre

Notre premier algorithme diviser pour régner ne fournit pas le bon résultat. Il nous reste en effet des paires de points à tester. Nous pouvons toutefois utiliser le résultat des appels récursifs pour limiter le nombre de paires à tester. Imaginons qu'après les appels récursifs, la paire de points les plus proches obtenue corresponde à une distance d_{rec}.

Question 10 – Soit $\mathbf{p}_1$ tel que $\mathbf{p}_1.x \leq x_{mid}$ et $\mathbf{p}_2$ tel que $\mathbf{p}_2.x > x_{mid}$. Si $d(\mathbf{p}_1, \mathbf{p}_2) < d_{rec}$, quelle relation doivent vérifier les abscisses (x) de $\mathbf{p}_1$ et $\mathbf{p}_2$ par rapport à x_{mid} ? Dans la suite, nous appelerons ces points les points centraux.

Soit tab_1 le tableaux des points centraux $\mathbf{p}$ tels que $\mathbf{p}.x \leq x_{mid}$ et tab_2 le tableau des points centraux $\mathbf{p}$ tels que $\mathbf{p}.x > x_{mid}$. Posons également que tab_1 et tab_2 sont triés par ordre de y croissant. Nous proposons l'algorithme suivant pour tester les paires de points centraux :

```
Fonction CompareCentre (tab1, tab2, d_rec) → (point, point) ou ∅
    d_min ← d_rec
    résultat ← ∅

    t1 ← taille de tab1
    t2 ← taille de tab2
    // i2 est l'indice du début de la zone de tab2 à tester pour les points de tab1
    i2 ← 0

    // pour chaque point de tab1
    pour i1 de 0 à t1 − 1 faire
        // faire monter i2 pour éliminer des points trop bas de tab2 pour la suite
        tant que i2 < t2 et tab2[i2].y + d_rec < tab1[i1].y faire
            i2 ← i2 + 1
        // à partir de i2 remonter tab2 jusqu'à être trop haut par rapport au point de tab1 testé
        j2 ← i2
        tant que j2 < t2 et tab2[j2].y − d_rec < tab1[i1].y faire
            // tester si la paire de points diminue la distance minimale connue
            si d(tab1[i1], tab2[j2]) < d_min alors
                d_min ← d(tab1[i1], tab2[j2])
                résultat ← (tab1[i1], tab2[j2])
            j2 ← j2 + 1
    retourner résultat
```

Question 11 – Pour les tableaux tab_1 et tab_2 suivants, avec $d_{rec} = 3$, listez les paires de points qui seront comparées, dans l'ordre dans lequel elles sont réalisées. Vous ne vous préoccuperez pas de la mise à jour de d_{min} et du résultat.

	0	1	2	3
tab_1 :	(0,0)	(1,2)	(0,6)	(1,10)

	0	1	2	3	4	5
tab_2 :	(3,1)	(4,3)	(2,4)	(3,6)	(3,9)	(4,11)

Question 12 – Quelle est la complexité de cet algorithme dans le meilleur et le pire des cas, en fonction des tailles t_1 et t_2 de tab_1 et tab_2 ? Illustrez chaque cas d'un dessin.

D'après nos appels récursifs, nous savons que la distance minimale entre deux points de tab_2 est d_{rec}. Avec cette information, il est en réalité possible de montrer que pour chaque point de tab_1, notre algorithme teste au maximum les distances avec 6 points de tab_2.

Question 13 – Quelle est dans ce cas la complexité de l'algorithme CompareCentre ?

Pour utiliser CompareCentre, nous avons besoin des points triés par y croissant. Supposons que nous disposions des points dans deux tableaux : xtab contient les points triés par x croissant et ytab contient les mêmes points triés par y croissant. Supposons également que les points ont tous des x deux à deux distincts.

Question 14 – Soit milieu l'indice d'un point dans xtab. Écrivez un algorithme permettant de séparer le tableau ytab en deux nouveaux tableaux yinf et ysup également triés par y croissant, de telle sorte que tous les points de xtab avant milieu (exclus) soient dans yinf et tous les points après milieu (inclus) soient dans ysup. Détaillez sa complexité en fonction du nombre de points.

Question 15 – Votre algorithme fonctionne-t-il toujours lorsque les points peuvent avoir des x identiques ? Détaillez bien votre réponse et fournissez des exemples.

6 Algorithme final (pas plus de questions ici)

Muni de toutes les briques que nous avons évoquées, nous pouvons désormais formuler l'algorithme final. S'il vous reste du temps, vous pouvez détailler la complexité de l'ensemble en bonus.

Fonction PlusProchesRec (xtab, ytab, début, fin) → (point, point)
- // arrêt
- **si** fin- début < 2 **alors**
  - Erreur : trop peu de points
- **si** fin- début ≤ 3 **alors**
  - tester toutes les paires de points possibles pour les points de tab entre début (inclus) et fin (exclus)
  - **retourner** la paire de points les plus proches
- // division
- milieu ← (début + fin)/2
- partitionner ytab en yinf et ysup selon milieu (question 14)
- $(\mathbf{p}_1, \mathbf{p}_2)$ ← PlusProchesRec(xtab, yinf, début, milieu)
- $(\mathbf{q}_1, \mathbf{q}_2)$ ← PlusProchesRec(xtab, ysup, milieu, fin)
- // gestion du centre
- $d_{rec} \leftarrow \min(d(\mathbf{p}_1, \mathbf{p}_2), d(\mathbf{q}_1, \mathbf{q}_2))$
- tab_1 ← points centraux triés par y croissant extraits de yinf
- tab_2 ← points centraux triés par y croissant extraits de ysup
- rcentre ← CompareCentre(tab_1, tab_2, d_{rec})
- // résultat
- **si** rcentre $\neq \emptyset$ **alors**
  - **retourner** rcentre
- **si** $d(\mathbf{p}_1, \mathbf{p}_2) < d(\mathbf{q}_1, \mathbf{q}_2)$ **alors**
  - **retourner** $(\mathbf{p}_1, \mathbf{p}_2)$
- **sinon**
  - **retourner** $(\mathbf{q}_1, \mathbf{q}_2)$

Fonction PlusProches (tab) → (point, point)
- $t \leftarrow$ taille de tab
- xtab ← tab trié par ordre croissant de x
- ytab ← tab trié par ordre croissant de y
- **retourner** PlusProchesRec(xtab, ytab, 0, t)