Feuille de TD N^o 5 : Diviser pour régner, les points les plus proches

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 $(M_0, M_1, ..., M_{n-1})$ dans un ordre quelconque pour le moment. Ils seront représentés en C par deux tableaux d'entiers 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 `xi=coords_x[i]` et pour ordonnée `yi=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.

La question 7 est un peu technique mais quitte à bien faire l'effort de lire le sujet et de s'en imprégner tout le reste est assez facile.

1 Approche exhaustive

On utilise la distance euclidienne définie par $d(M,N) = \sqrt{(x_M - x_N)^2 + (y_M - y_N)^2}$.

1 En utilisant la fonction `sqrt` de la librairie `math`, écrire une fonction `double distance (int i, int j)` qui prend en entrée deux indices `i` et `j` et renvoie la distance euclidienne entre les points M_i et M_j.

2 Ecrire une fonction `void plus_proche()` qui affiche (à l'aide de la commande `printf`) un couple d'indices (i,j) tels que les points M_i et M_j sont les plus proches dans le nuage de points. On pourra se contenter d'une complexité quadratique pour répondre à cette question.

2 Quelques outils

Nous allons dans la troisième partie obtenir une meilleure stratégie mais nous allons commencer par écrire quelques fonctions utiles.

```
void swap(int tab[],int i, int j){
   int save = tab[i];
   tab[i]=tab[j];
   tab[j]=save;
}


void mystere (int tab[], int taille){
   for (int i=0;i<taille;i++){
      int pos =i;
      while(pos>0 && tab[pos]<tab[pos-1]){
         swap(tab,pos,pos-1);
         pos--;
      }
   }
}
```

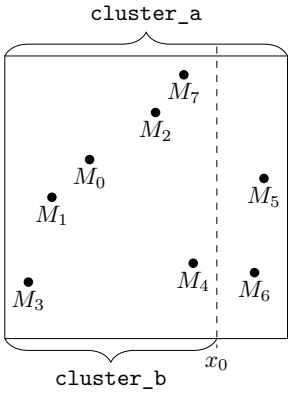
3 Que fait la fonction `mystere` ? Démontrer votre conjecture à l'aide d'un invariant de boucle ?

4 Comment s'appelle cet algorithme ? Quelle est sa complexité ?

5 On souhaite trier un tableau contenant des indices de points suivant l'ordre des abscisses croissantes. Quelles modifications faut-il apporter à la fonction précédente pour y parvenir ?

6 Citer un algorithme de tri par comparaisons dont la complexité est optimale.

On admettra que l'on dispose de deux tableaux de n entiers `liste_x` (resp. `liste_y`) 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 un tableau 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] ]
```

Pour que l'algorithme que nous allons concevoir dans la partie suivante soit efficace, il faut nous puissions éliminer des points du cluster efficacement (c'est-à-dire sans re-trier à chaque fois).

7 (question **un peu** technique que vous pouvez admettre pour la suite si nécessaire)
Ecrire une fonction `int** sous_cluster(int taille, int** clust, double x_min, double x_max)` qui renvoie un cluster (c'est-à-dire un tableaux à deux lignes) contenant les points du cluster passé en entrée dont l'abscisse est comprise entre `x_min` et `x_max`. La complexité devra être linéaire en la taille du cluster (variable `taille`).

8 Ecrire une fonction `int mediane(int taille, int** clust)` qui prend en entrée un cluster contenant au moins deux points et le nombre de points qu'il contient et renvoie une abscisse médiane c'est-à-dire telle qu'au moins la moitié des points a une abscisse inférieure ou égale et au moins la moitié en a une supérieure ou égale. La complexité sera constante.

3 Méthode "diviser pour régner"

Voici le principe de notre algorithme :

1. Si le cluster considéré contient deux ou trois points alors on calcule toutes les distances possibles pour répondre.
2. Si le cluster a au moins quatre points alors on le coupe en deux parties (égales à un point près) en coupant le cluster suivant la médiane des abscisses que l'on notera x_0. Notons D et G les parties du plan obtenues.
3. Il y a a alors trois situations possibles : les deux points les plus proches sont soit tous les deux dans G, soit tous les deux dans D, soit l'un est dans G et l'autre dans D. On va calculer récursivement les deux points les plus proches dans D ainsi que ceux dans G. Soit d_0 la plus petite distance obtenue par ces deux appels.
4. On cherche s'il existe une paire de points $M_1 \in D, M_2 \in G$ telle que $d(M_1, M_2) < d_0$. Si on trouve une telle paire, on renvoie sa distance et sinon on renvoie d_0.

Vois pouvez consulter la figure ci-dessous pour plus de détails :

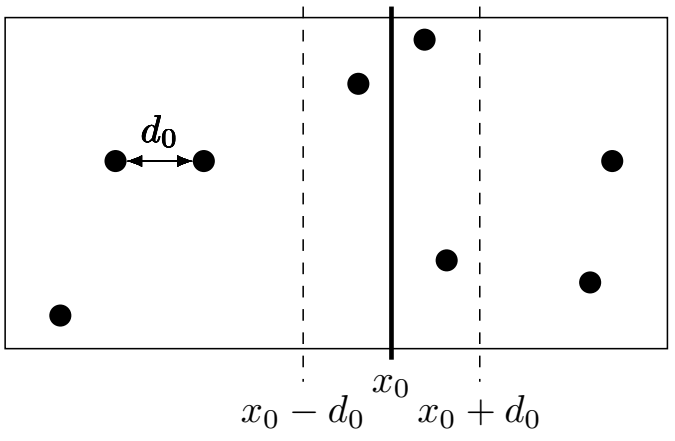


FIGURE 2 – Illustration du diviser pour régner

9 Ecrire une fonction `int** gauche(int taille, int** clust)` qui prend en entrée un cluster avec au moins deux points et renvoie le cluster constitué uniquement de la moitié (à un près) des points les plus à gauche du cluster passé en entrée.

On suppose qu'on dispose d'une fonction `int** droite (int taille, int** clust)` qui renvoie le cluster composé de la moitié des points les plus à droite. Cette fonction renvoie le complémentaire des points sélectionnés par la fonction `gauche`.

10 A l'étape 4, expliquer dans quel intervalle I_0 d'absisses on peut chercher le couple M_1, M_2.

Ecrire une fonction `int** bande_centrale(int taille, int** clust, double d0)` qui prend en entrée un cluster et un réel `d0` et qui renvoie le cluster composé des points du cluster passé en entrée dont l'abscisse est dans I_0. La complexité sera linéaire en la taille du cluster.

11 Soient deux points M_1 et M_2 situés à une distance inférieure stricte à d_0 comme dans l'étape 4 c'est-à-dire que l'un est dans G et l'autre dans D. Justifier que ces deux points se trouvent dans un rectangle de taille $2d_0 \times d_0$ à préciser (un dessin suffira). Justifier qu'un tel rectangle ne peut pas contenir plus de 8 points du problème. Ainsi les points M_1 et M_2 se trouvent, dans la deuxième ligne du cluster (celle triée par ordonnées croissantes), séparés d'au plus 7 éléments.

On pourra montrer par l'absurde qu'un rectangle à déterminer de taille $2d_0 \times d_0$ contient au plus 8 points. (faire un dessin)

12 En déduire une fonction `double fusion(int taille,int** clust, double d0)` qui prend en entrée un cluster dont tous les points ont leur abscisse dans $[x_0 - d_0, x_0 + d_0]$ et sa taille et renvoie la distance minimale entre deux points du cluster si elle est inférieure à d_0 ou d_0 sinon. La complexité de cette fonction sera linéaire en la taille du cluster : le justifier.

13 Ecrire une fonction `double distance_minimale(int taille, int** clust)` qui prend en entrée un cluster et renvoie la distance minimale entre deux points qu'il contient à l'aide de la stratégie diviser pour régner.

Nous admettrons (pour simplifier) pour cette question que l'on dispose d'une fonction `taillec` qui permet d'obtenir la taille d'un cluster.

14 Si on note n la taille du cluster et $C(n)$ le nombre d'opérations élémentaires réalisées par la fonction `distance_minimale`, justifier que l'on a :

$$C(n) = 2C(n/2) + O(n)$$

15 En déduire, en la démontrant, la complexité $C(n)$ quand $\exists k \in \mathbb{N}, n = 2^k$.