Université Paris Dauphine
IUP Génie Mathématique et Informatique
IUP1 – Année 2001-2002
Algorithmique et structures de données

Devoir à rendre le 12 fevrier 2002
Partitionnement de points par l'algorithme de la médiane

Comme d'habitude, chaque question sera accompagnée du listing du code C correspondant, ainsi que du listing du résultat de l'exécution des programmes demandés.

Soit `G` un ensemble de `g` points dans un espace de dimension `N`. Le but du devoir est de programmer un algorithme qui, étant donné un nombre `p`, construit un ensemble `P` de `p` points qui *représentent bien* `G`. On supposera de plus que `p` est une puissance de 2 : $\mathtt{p} = 2^{\mathtt{q}}$.

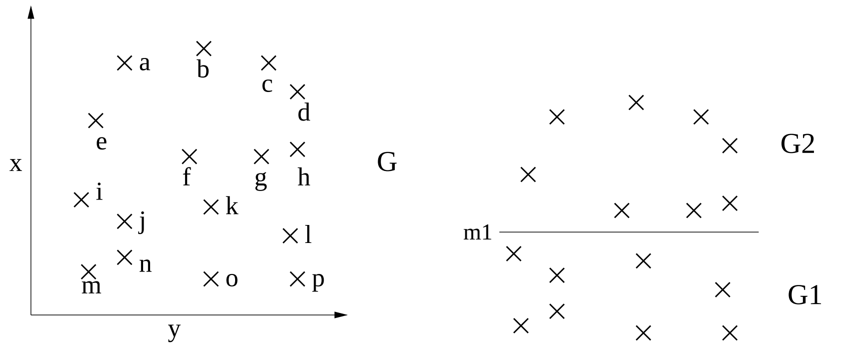
L'algorithme fonctionne en deux temps :

1. on partitionne l'espace en `p` parties de manière à ce que chaque partie contienne à peu près le même nombre d'éléments de `G` (entre $\lfloor \mathtt{g/p} \rfloor$ et $\lceil \mathtt{g/p} \rceil$) ;
2. chaque élément de `P` est le barycentre (moyenne composante à composante) des points de `G` contenus dans une des `p` parties de la partition.

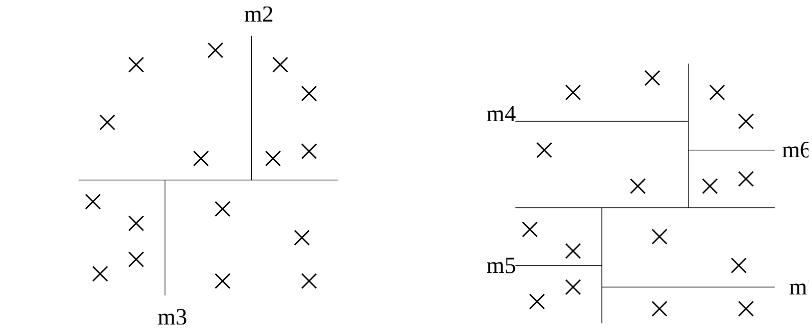
Le calcul d'une partition adéquate se fait en coupant récursivement l'espace en deux, par un plan perpendiculaire à un axe ; on change d'axe à chaque découpage récursif :

1. on cherche une valeur médiane `m` telle qu'il y ait autant de points de `G` dont la `k`-ième composante soit plus grande que `m`, que de points dont la `k`-ième composante soit plus petite que `m` ;
2. on partitionne `G` en $\mathtt{G}_1$ et $\mathtt{G}_2$ contenant les points de `G` dont la `k`-ième composante est, respectivement, plus petite ou plus grande que `m` ;
3. on applique récursivement l'algorithme à $\mathtt{G}_1$ et à $\mathtt{G}_2$, mais par rapport à la `k+1`-ième composante (modulo `N`), et ainsi de suite.

Au bout de `q` itérations, on a partitionné l'ensemble initial `G` en $2^{\mathtt{q}} = \mathtt{p}$ parties d'à peu près même cardinal.
Exemple : partition de 16 points en 8 parties, en dimension 2 :

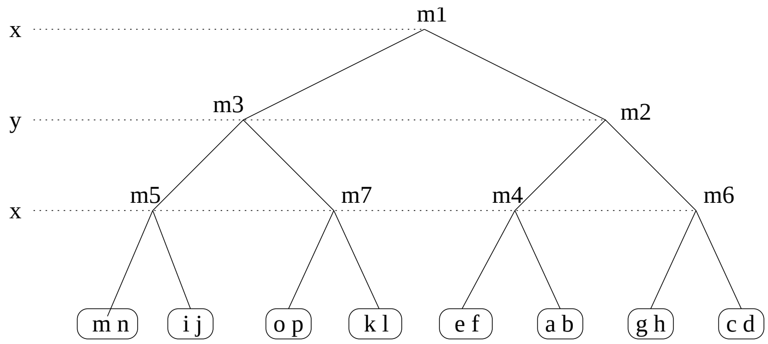


(a) ensemble `G` original (b) première coupure, selon l'axe `x`



(c) deuxième coupure, selon l'axe `y` (d) dernière coupure, selon l'axe `x`

Cette partition est naturellement représentée par un arbre binaire complet de hauteur `q`. À chaque niveau est associée la composante `k` selon laquelle a été fait le découpage. À chaque nœud interne est associée la médiane `m` utilisée pour partitionner selon la `k`-ième composante. Chaque feuille correspond à une partie de la partition ; on stocke donc dans chaque feuille le sous-ensemble de `G` contenu dans la partie associée.



Question 1 : calcul de la médiane.

Dans toutes la suite, on suppose qu'une constante symbolique `N` définit le nombre de dimensions de l'espace et que `pt` est le type des points dans cet espace ; par exemple, en dimension 2 :

```
#define N 2
typedef double pt[N] ;
```

Programmez la fonction

```
double mediane(pt* p, int n, int k) ;
```

qui prend en entrée un tableau `p` de `n` points et renvoie une médiane `m` des `k`-ième composantes de `p`. On doit avoir $\forall i < \mathtt{n}/2,\ \mathtt{p}[i][\mathtt{k}] \leq \mathtt{m}$ et $\forall i > \mathtt{n}/2,\ \mathtt{p}[i][\mathtt{k}] \geq \mathtt{m}$.
Une méthode possible est de trier le tableau `p`, puis de choisir `m=p[n/2][k]`.
Que vous utilisiez ou non un tri, vous expliquerez votre algorithme et donnerez sa complexité. Vous ferez également un petit programme d'exemple qui utilise la fonction `mediane` pour calculer et imprimer sur l'écran la médiane du tableau unidimensionnel `{ 100,12,14,98,24,18,2 }`.

Question 2 : calcul de la partition.

La partition est représentée par un arbre binaire complet utilisant des cellules de type `node_t` :

```
typedef struct node_tt {
  struct node_tt *left,*right ;   /* fils gauche et droit */
  int k ;                         /* composante servant à séparer les points */
  double m ;                      /* médiane sur la composante k */
  pt* G ;                         /* ensemble des points contenus dans la partie */
  int g ;                         /* taille du tableau G */
} node_t ;
```

Le champs `G` et `g` ne sont utilisés que quand le nœud est une feuille ; les champs `k` et `m` ne sont utilisés que quand le nœud est un nœud interne.

Programmez la fonction

```
node_t* partition(pt* G, int g, int q) ;
```

qui construit l'arbre binaire complet de hauteur `q` correspondant à la partition des `g` points du tableau `G` en $2^{\mathtt{q}}$ parties. Cette fonction pourra faire appel à une fonction auxiliaire récursive qui prend en argument un tableau `T` de points, détermine sa médiane grâce à la fonction `mediane`, découpe `T` en deux tableaux `T1` et `T2`, s'appelle récursivement sur `T1` et `T2` pour obtenir deux arbres `A1` et `A2`, et renvoie l'arbre construit en liant `A1` et `A2` par une nouvelle racine.

Programmez la fonction

```
pt* quantifie(node_t* A, int q) ;
```

qui prend l'arbre de partition `A` et renvoie un tableau de $2^{\mathtt{q}}$ points correspondant au barycentre de chaque partie.

Question 3 : application.

Soit dans le plan (donc $\mathtt{N} = 2$) l'ensemble de 10000 points $\mathtt{G} = \{\ (i, i^2) \mid 0 \leq i < 10000\ \}$. Faites un petit programme d'exemple qui calcule et imprime sur l'écran l'ensemble `P` pour $\mathtt{p} = 16$.