Les deux points les plus proches

Tahina Ramananandro
ramanana@clipper.ens.fr

Lycée Louis-le-Grand – MPSI CAML
7 mai 2008

Étant donné un ensemble fini de points du plan, on veut déterminer parmi ces points le couple des deux points (distincts) les plus proches (au sens de la distance euclidienne).

Chaque point est représenté par le couple de son abscisse et de son ordonnée :

```
type point == (float * float);;
```

L'ensemble des points considérés est vu comme un tableau (`point vect`). **Dans toute la suite, on supposera que les abscisses des points sont toutes distinctes deux à deux.**

Ne pas oublier de tester les fonctions aussitôt écrites.

1 Méthode naïve

Écrire une fonction :

```
naive : point vect -> ((point * point) * float)
```

qui parcourt tout le tableau et retourne les deux points, ainsi que leur distance, en examinant tous les couples de points distincts possibles.

Combien d'additions et de multiplications sont nécessaires à ce calcul ? Combien de comparaisons ?

2 Diviser pour régner

Nous allons examiner une méthode plus fine, qui consiste à :

1. Séparer le plan en deux suivant une droite verticale,
2. Calculer récursivement le couple des deux points les plus proches dans chaque demi-plan récursivement,
3. Comparer les couples obtenus avec le couple des deux points les plus proches de la droite de séparation de part et d'autre.

2.1 Préparation

On veut d'abord construire :

– la liste des points triés par ordonnées croissantes,
– le tableau des points triés par abscisses croissantes.

Programmer un tri (de préférence le moins complexe possible) pour construire ces deux objets et donner sa complexité. On pourra éventuellement utiliser les fonctions CAML list_of_vect et vect_of_list (mais à bon escient !)

2.2 Séparation du plan

Écrire une fonction :

```
separer : point vect -> int -> int -> point list -> (point list * point list)
```

telle que, si :

– trie_par_X est le tableau de tous les points triés par abscisses croissantes
– trie_par_Y_partiel est la liste, triée par ordonnées croissantes, des points de trie_par_X à partir de l'indice i et jusqu'à l'indice j,

alors `separer trie_par_X i j trie_par_Y_partiel` construit les deux listes, triées par ordonnées croissantes, des points de `trie_par_X`, d'une part entre les indices i et $\frac{i+j}{2}$, d'autre part entre les indices $\frac{i+j}{2} + 1$ et j.

Quel est son nombre de comparaisons en fonction de i et j ?

2.3 Couples à cheval sur la ligne de séparation

Supposons qu'on ait déjà calculé chacun des couples les plus proches de part et d'autre de la ligne de séparation. Soit *dist* leur distance.

Construire d'abord une fonction :

`a_cheval : point vect -> int -> int -> point list -> float -> point list`

telle que `a_cheval trie_par_X i j trie_par_Y_partiel dist` renvoie la liste, triée par ordonnées croissantes, des points de `trie_par_X` entre les indices i et j et situés à une distance d'au plus *dist* de la droite de séparation (dont l'abscisse est `fst trie_par_X.(m)`, l'abscisse du point d'indice m dans `trie_par_X` avec $m = \frac{i+j}{2}$).

Étant donné un point P de cette liste, quel est le nombre maximum de points de cette liste dont l'ordonnée dépasse d'au plus *dist* celle de P ?

À partir de cette remarque, construire une fonction qui extrait de cette liste le couple des deux points les plus proches de cette liste.

2.4 Résumé

Écrire le programme final :

`les_plus_proches : point vect -> ((point * point) * float)`

qui à partir d'un tableau quelconque de points du plan, calcule le couple des points les plus proches ainsi que leur distance. Quelle est sa complexité en termes de nombre d'additions, multiplications ? En termes de nombre de comparaisons ?

3 Partons dans l'espace

Comment généraliser cette méthode à l'espace ? Avec quelle complexité ?