

TECHNIQUES ALGORITHMIQUES ET PROGRAMMATION

Paire de points les plus proches

On va étudier un algorithme qui prend en entrée un ensemble de points P du plan et retourne la paire de points les plus proches, et qui répond rapidement en moyenne.

Étant donné un réel $\delta > 0$, la cellule d'indice (i, j) , où i, j sont entiers, est la région carrée du plan de côté δ correspondant à $[i\delta, (i+1)\delta[\times [j\delta, (j+1)\delta[$. Notez que les cellules partitionnent le plan.

Dans l'algorithme qui suit, on visite chaque point de P dans l'ordre $p_0, p_1, \dots, p_t, \dots$. À l'étape t , on aura besoin de déterminer rapidement si une cellule (i, j) contient un des points déjà visités. On va le faire grâce à une table de hachage T_δ indexée par les indices (i, j) des cellules et contenant une liste de points. Donc $T_\delta[(i, j)] = \{p_{t_1}, p_{t_2}, \dots\}$ représente l'ensemble courant (éventuellement vide) des points déjà rencontrés et appartenant à la cellule (i, j) .

Algorithme `pppp_random(P)`

Entrée : Une suite $P = (p_0, p_1, \dots, p_{n-1})$ de $n \geq 2$ points du plan.

Sortie : La paire de points les plus proches.

1. Poser $d_{\min} := d(p_0, p_1)$ et $R := \{p_0, p_1\}$.
 2. Créer une table de hachage T_δ de cellules de paramètre $\delta := d_{\min}/\sqrt{2}$.
 3. Pour chaque point $p_t \in P$ dans l'ordre $t := 0, 1, \dots, n-1$:
 - (a) Soit $(i, j) := (\lfloor x(p_t)/\delta \rfloor, \lfloor y(p_t)/\delta \rfloor)$ la cellule du point p_t .
 - (b) Chercher dans la table T_δ le point q qui, parmi les cellules "voisines" de (i, j) , soit le plus proche de p_t .
 - (c) Si q existe¹ et si $d(p_t, q) < d_{\min}$, poser $d_{\min} := d(p_t, q)$, $R := \{p_t, q\}$ et aller en (2).
 - (d) Ajouter p_t à $T_\delta[(i, j)]$.
 4. Renvoyer R .
-

Question 1. Montrez qu'à chaque instant de l'algorithme, $T_\delta[(i, j)]$ contient au plus un seul élément.

Question 2. L'algorithme ne précise pas en (3b) ce qu'est une cellule "voisine" de (i, j) . Pouvez-vous préciser ? Combien de cellules faut-il considérer au maximum ? au minimum ?

Question 3. Montrez que si la boucle en 3 termine, c'est-à-dire que le test en (3c) n'est jamais vrai, alors les deux points de P les plus proches sont à distance $\geq d_{\min}$. En déduire que l'algorithme `pppp_random(P)` calcule dans R la paire de points la plus proche.

On supposera qu'on dispose d'une implémentation de table de hachage permettant la création, la lecture et l'insertion en temps constant (en moyenne).

Question 4. Pourquoi n'est-il pas judicieux d'utiliser, à la place d'une table de hachage $T_\delta[(i, j)]$, un simple tableau à deux dimensions $T_\delta[i][j]$?

1. Toutes les cellules "voisines" de (i, j) pourraient être vides.

On dira qu'il y a *échec* à l'indice t si le test en (3c) devient vrai pour p_t . Le coût total de l'algorithme, c'est-à-dire sa complexité en temps, peut donc se décomposer en une suite d'échecs jusqu'au moment où la boucle en 3 se termine sans échec.

Question 5. *Quel est le coût de la boucle en 3 lorsqu'il y a échec à l'indice t ?*

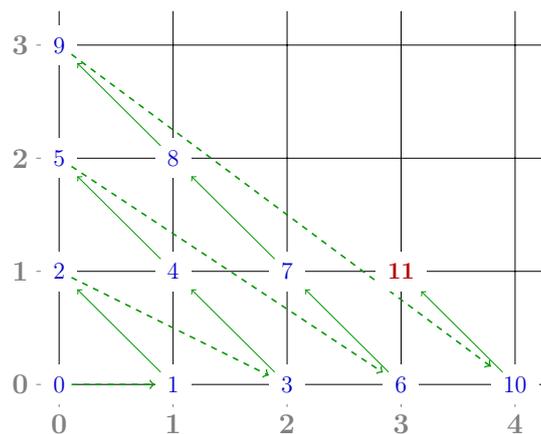
Question 6. *Montrez que s'il y a un échec à l'indice t puis un échec à l'indice t' , alors $t' > t$. En déduire la complexité de `pppp_random(P)` dans le pire des cas. Décrire une suite de points qui atteint ce pire cas.*

Question 7. *Montrez que si les points sont permutés aléatoirement uniformément dans P , alors la probabilité d'avoir un échec à l'indice t est au plus la probabilité que p_t réalise la plus petite distance entre deux points d'indice $\leq t$. En déduire que la probabilité d'avoir un échec à l'indice t est $\leq t / \binom{t+1}{2}$.*

Question 8. *En déduire que la complexité moyenne de l'algorithme qui commencerait par permuter aléatoirement les points et qui appliquerait `pppp_random(P)` est en $O(n)$.*

Pour implémenter T_δ on aura besoin de coder chaque couple d'entiers $(i, j) \in \mathbb{N}^2$ vers un entier unique noté $\text{map}(i, j)$. En représentant chaque couple (i, j) par un point d'une grille, on définit alors $\text{map}(i, j)$ comme la somme des longueurs des diagonales du type $(t, 0) \rightarrow (0, t)$, partant de $(0, 0)$ et jusqu'à atteindre le point (i, j) sans le compter (voir la figure). Ici, la longueur d'une diagonale est le nombre de points de la grille qu'elle contient. On aurait pu prendre $2^i \cdot 3^j$, mais c'est plus coûteux à calculer.

Dans l'exemple ci-dessous, $\text{map}(3, 1) = 11$ car il y a exactement 11 points sur les diagonales situées avant $(3, 1)$. Notez que la dernière diagonale qui contient (i, j) démarre à l'abscisse $i + j = 4$.



Question 9. *Montrez que $\text{map}(i, j) = \left(\sum_{t=0}^{i+j-1} (t+1) \right) + j$, et en déduire une formule close.*

En TP

Télécharger les fichiers correspondant au TP à partir de la page de l'UE disponible ci-après :

<http://dept-info.labri.fr/~gavoille/UE-TAP/>

Complétez le fichier `pppp.c` afin d'implémenter les trois algorithmes (fonctions) `pppp_naive()`, `pppp_divide()` et `pppp_random()` décrites dans `pppp.h`.

Pour ce TP, on va réutiliser `tsp_main.c` qui va permettre de visualiser la paire de points la plus proche, de tester les fonctions sur des exemples particuliers, etc. Dans `tsp_main.c`, il faudra :

- décommenter `#include "tsp_brute_force.h"` pour la fonction de distance `dist()`
- décommenter `#include "tsp_mst.h"` (pour `createGraph`) et bien sûr `#include "pppp.h"`
- désactiver les parties liées au TSP, notamment `#ifdef TSP_BRUTE_FORCE_H` et `#ifdef TSP_MST_H`
- commenter `#include "tsp_prog_dyn.h"` et `#include "tsp_heuristic.h"`