Compte rendu TP : Arboretum - Exercice 3

Mohammed Amine KHELDOUNI

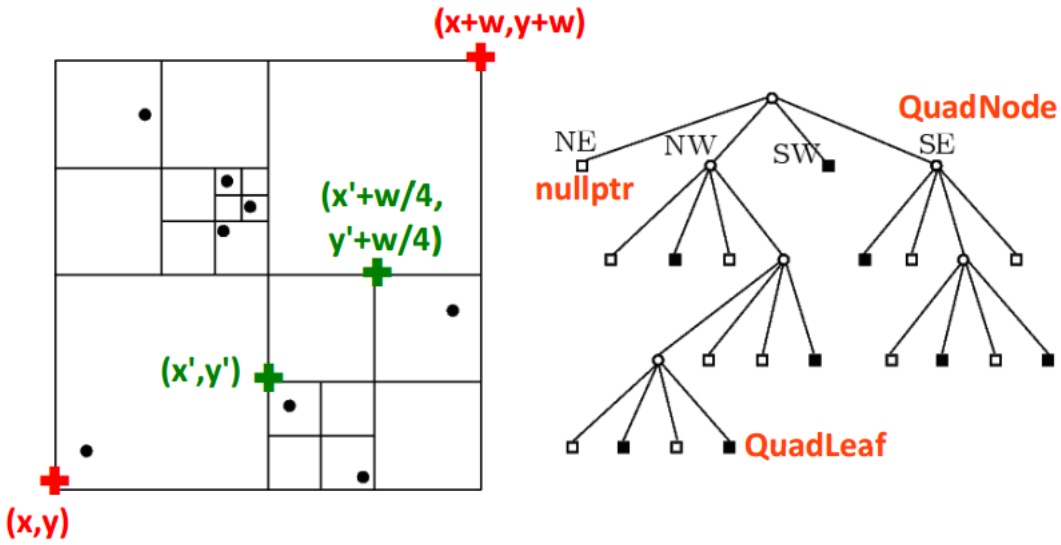
5 décembre 2016

Introduction

On se propose d'étudier les propriétés des **quadtrees** ou **arbres quaternaires** qui constituent une structure de donnée de type arbre dans laquelle chaque noeud a quatre fils.
On utilise ici un quadtree pour partitionner un espace bidimensionnel (une carte de villes dont les positions sont projetées selon les coordonnées de Lambert) en le subdivisant récursivement en quatre quadtrees.
Le quadtree nous permettera alors d'étudier les propriétés géographiques des villes et plus spécialement les proximités des villes entre elles. En effet, nous proposons dans ce rapport un algorithme de recherche de plus proche voisin, vu en cours, qui permet de trouver le voisin le plus proche à une ville donnée.

Création du QuadTree

Pour notre problème, on privilégie l'usage de quadtree "point" qui permet de représenter une coordonnée géographique.
Ainsi, le quadtree est un arbre dont chaque noeud possède quatre fils, possédant chacun une direction. Géographiquement, les directions sont notées : NW, NE, SE et SW.
Enfin, chaque feuille est soit vide, soit possède un point. C'est à dire que chaque feuille peut avoir ou non une ville dans sa localisation.
On peut schématiser le quadtree utilisé comme suit :



Pour mieux diviser le quadtree en branches, on introduit les classes "Square" et "Quadrant". Ainsi, on effectue pour un carré (une région) connu par la donnée de sa position (x, y) et de sa largeur w, une décomposition en quatre sous carrés qui s'associent chacun à une des directions du quadtree. Chaque entité comportant un sous carré et une direction relative au carré initial s'appelle un "Quadrant". Donc comme le montre le schéma ci-dessus, on peut décomposer le carré initial en plein de carrés jusqu'à isoler chaque point de notre quadtree dans un propre sous carré.

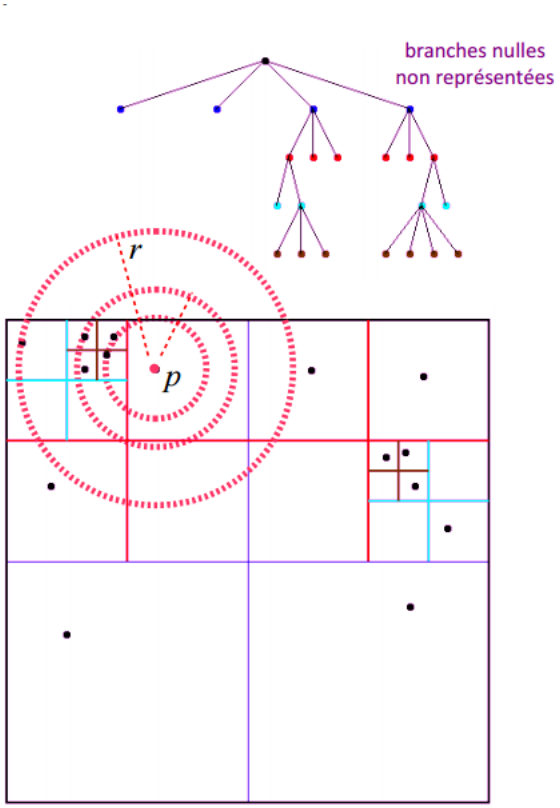
Etant donné notre quadtree Q, on insère un point P dans Q connaissant notre carré initial S en effectuant les étapes suivantes (cf. void inser dans neighbors.h) :

- Si le quadtree est vide, on y insère une feuille avec comme valeur le point P
- Si Q est un noeud, il suffit de réitérer la fonction sur le sous carré et le fils du noeud Q qui a la bonne direction (celle du Quadrant auquel P appartient)
- Si Q est une feuille et qu'il possède déjà un point, il faut le transformer en noeud pour le diviser afin de séparer le point déjà existant dans la feuille et P, le point qu'on veut insérer.

Implémentation de recherche du plus proche voisin

Après avoir formé notre quadtree avec les points associés à nos villes (coordonnées des villes), on cherche un algorithme qui permette de trouver le plus proche voisin d'un point donné.
Pour cela, il suffit de prendre un disque de centre le point P de départ et de rayon r. Ce rayon augmente petit à petit avec notre test d'intersection disque-carré et stocke dans un vecteur de voisins ceux qui valident le test.
Ainsi, on propose une implémentation de l'algorithme du plus proche voisin vu dans le cours sur les arboretum en suivant les étapes (cf. neighbors.h et quadrant.cpp) :

- Si le quadtree est vide, alors on a 0 voisins.
- Si le quadtree Q est une feuille, on calcule la distance du point P à la feuille (dont la valeur est un point P'), si elle est plus petite que r^2, on enregistre le voisin.
- Si Q est un noeud, on le subdivise en quatre quadtrees, et on recalcule les distances des sous quadtree par rapport au point P si le sous quadtree est une feuille, sinon on vérifie l'intersection entre le sous carré associé au sous quadtree i avec le disque $D(p, r)$. S'ils s'intéresectent, on réitère la fonction pour notre sous quadtree.



Pour la ville **Ponts** on trouve que la ville la plus proche est **Saint-Jean-de-la-Haize**.

Interprétation des résultats et des temps de calcul

Passons maintenant aux interprétations :

On charge pour notre application les villes de France métropolitaine données dans le fichier "villes.txt" et on les insérent dans le quadtree de départ grâce à une fonction "create$_q tree$".

Taille du quadtree

On trouve un quadtree d'une taille de :

$$nombre_{leaves} + nombre_{nodes} = 55037$$

Avec une profondeur maximale de (cf. fonction depth dans quatree.h) :

$$depth = 16$$

Ville la plus proche de "Ponts"

Ainsi, on trouve qu'il faut parcourir 139 villes du quadtree Q avant de trouver le plus proche voisin de la ville "Ponts" qui est "Saint-Jean-de-la-Haize".
Le nombre de ville parcourues est négligeable par rapport à un parcours linéaire, en effet :

$$\frac{140}{55040} \approx 2.54 * 10^{-3}$$

Donc cet algorithme est mille fois plus rapide qu'un parcours linéaire de la liste des villes.

Temps moyen

On compare ici les temps moyens pour trouver une plus proche ville avec un quadtree ou avec un algorithme naif linéaire qui parcourt la liste des villes (cf example.cpp) :

$$\tau_{qtree} \approx 3.8 * 10^{-4}\, s$$

$$\tau_{linear} \approx 1.5 * 10^{-3}\, s$$

D'où un temps moyen plus rapide pour le quadtree, sur une moyenne d'itération d'environ mille villes.

Temps de construction du QuadTree

On veut maintenant rentabiliser le temps de construction du quadtree égale, selon le code joint au rapport, à :

$$\tau_{buildQTree} = 5.7 * 10^{-2}\, s$$

Ainsi, on voudrait que notre temps total de création du quadtree additionné au temps de recherche du plus proche voisin soit plus petit que le temps en algorithme linéaire. On cherche donc x le plus petit possible tel que :

$$\tau_{buildQTree} + x\tau_{qtree} \leq x\tau_{linear}$$

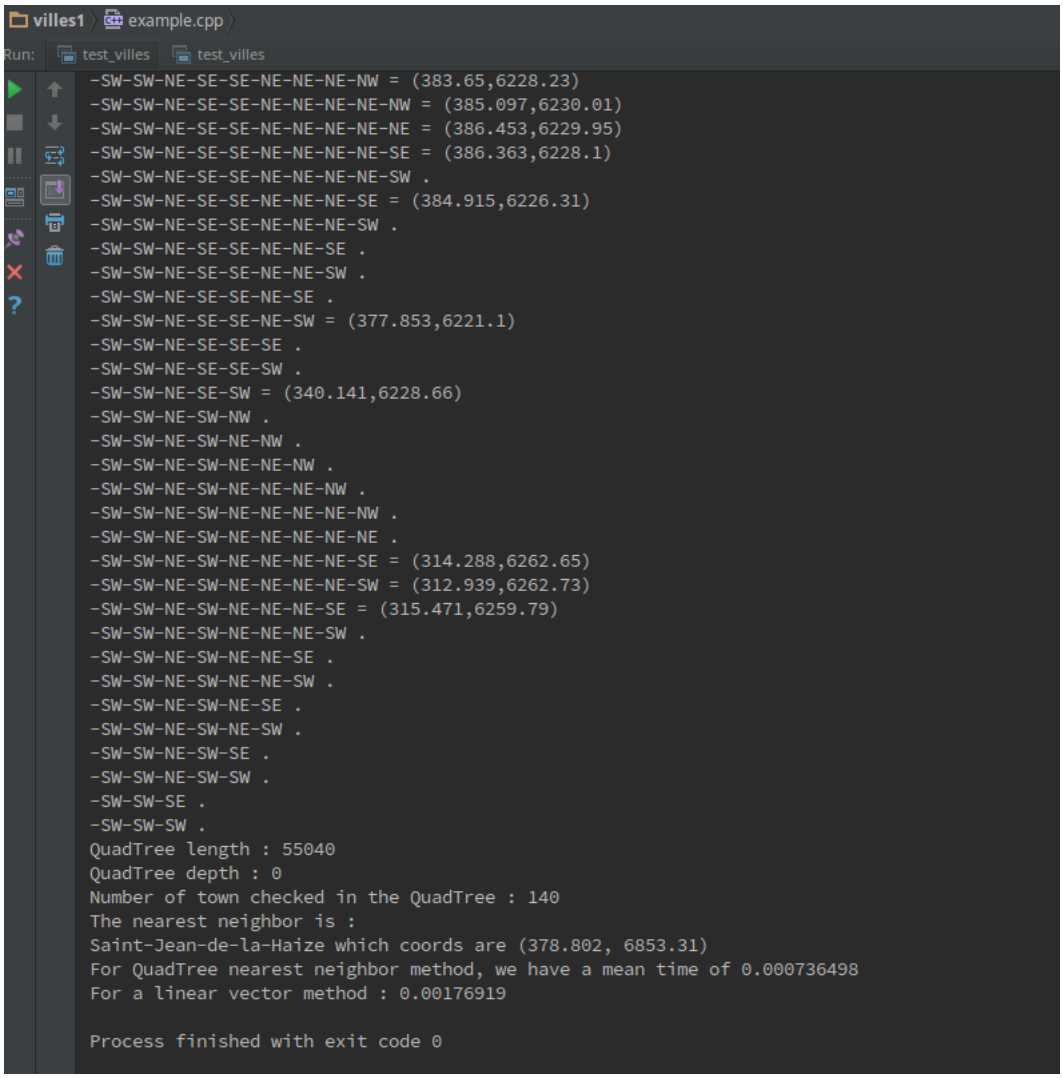
Ce qui est vérifié à partir d'environ 67 recherches de plus proche voisin.

```
/home/amine/.CLion2016.2/system/cmake/generated/villes1-3405dc79/3405dc79/Debug/test_villes
Reading town file: /home/amine/Documents/2A/Prog Av/TP5/villes1/villes.txt
File read in 0.068533 s
Number of towns in file: 35181
Bounding box: (102.504,6052.39)-(1239.5,7109.86)
Number of towns with this name: 1
  (lat=48.7, lon=-1.35)
Looking for nearest neighbor of : Ponts
Our town's coords are (380.027, 6853.24)
Time taken to build the quadtree 0.050026
QuadTree length : 55040
QuadTree depth : 0
Number of town checked in the QuadTree : 140
The nearest neighbor is :
Saint-Jean-de-la-Haize which coords are (378.802, 6853.31)
For QuadTree nearest neighbor method, we have a mean time of 0.000392465
For a linear vector method : 0.00134429

Process finished with exit code 0
```

Conclusion

On a ainsi réussi à mettre en évidence les propriétés remarquables des QuadTree et leur efficacité à résoudre certains problèmes comme celui du plus proche voisin vu dans ce TP.
Voici pour finir un affichage du QuadTree avec ses propriétés, et le calcul des temps moyens.



Références

- Slides du cours de Renaud MARLET, ENPC - MOPSI (2016-2017)

- Discussion sur le TP - Exercice 3 avec Jean-Christophe CORVISIER et Yonatan DELORO, et sur l'exercice 2 avec Victor MARCHAIS

- https://fr.wikipedia.org/wiki/Quadtree

- http://stackoverflow.com/questions/401847/circle-rectangle-collision-detection-intersection