17 Janvier 2024

TP Java

Louis Delyon

Table des matières

I Introduction

Un **Quadtree** ou arbre quaternaire est une structure de données de type arbre dans laquelle chaque nœud a quatre fils. Les quadtrees sont le plus souvent utilisés pour partitionner un espace bidimensionnel en le subdivisant récursivement en quatre nœuds.

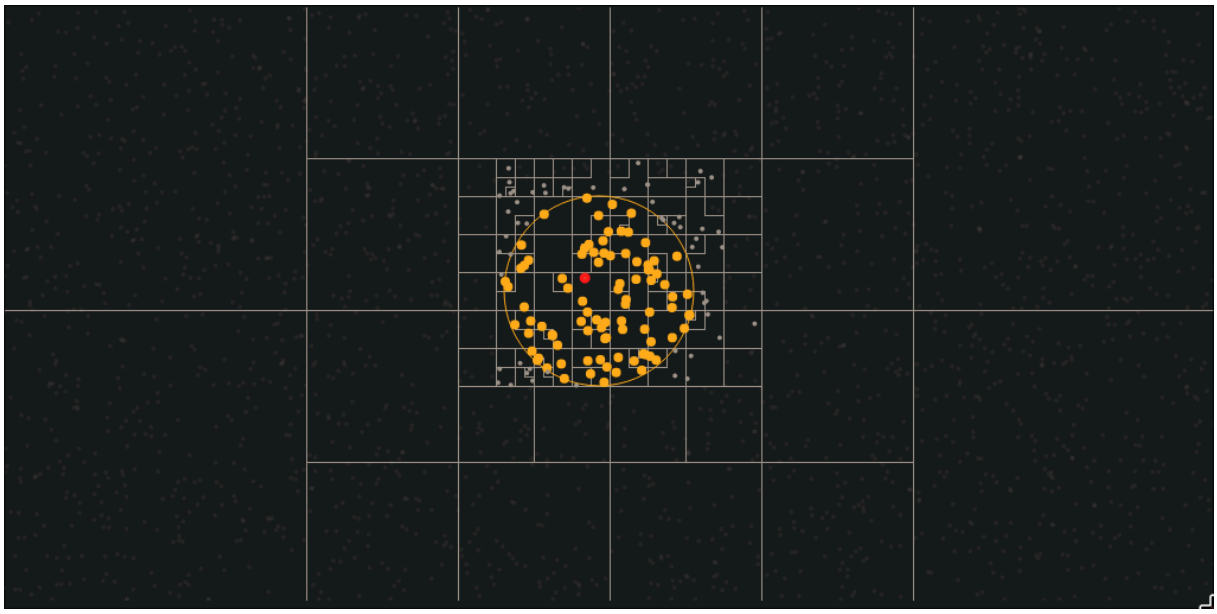
Cette structure de données peut être utilisée à diverses fins, par exemple pour de la compression d'images (regroupant sous un seul bloc les grosses zones d'une même couleur). L'utilisation qui nous interesse dans ce TP est celle permettant l'optimisation de recherche de proximité.

I.1 Énnoncé du problème

Soit un nuage de points dans un espace bidimensionnel trouver tous les points dans un rayon autour des coordonnées en entrée, en précisant le point le plus proche.

Exemple d'application : étant donné la liste des restaurants dans le monde, trouver tous les réstaurants à moins de x km, et donner le réstaurant le plus proche.

Pour simplifier, on va supposer que toutes les coordonnées sont des réels entre -1 et 1.



II Recherche naïve

On va ici implémenter une methode « naïve » de résolution de ce problème, on se propose de stocker le nuage de points sous forme d'une liste de coordonnées.

1. Écrire une classe `Point` représentant un point dans un espace 2D.
2. Écrivez le constructeur de la classe `Point`, ce constructeur doit renvoyer une erreur si les coordonnées ne sont pas comprises entre -1 et 1
3. Ajouter une methode `dist` à la class `Point` calculant la distance entre deux points.
4. Proposer une structure de données simple (pas encore de Quadtree) adaptée à ce problème, pourquoi ? Donner son type en Java.
5. Écrivez une méthode statique de la classe `Main` prenant en entrée une position, une distance et un nuage de points (comme décrit à la question précédente) et renvoyant la liste des points du nuages à une distance inferieure à la distance d'entrée.

Cette fonction parcours tout simplement le nuage de points en calculant la distance entre chaque point du nuage et la position

Pour ceci, vous pouvez créer une classe générique `Pair<U,V>`

III Les Quadtrees

III.1 Création des objets

Commençons par définir nos objets :

1. Créez:
   - Une classe abstraite `QuadTree` ayant comme *attributs* 4 `doubles` `xmin`, `ymin`, `xmax`, `ymax` correspondant aux coordonnées des points en haut à gauche et en bas à droite
   - Une classe `QuadNode` héritant de `QuadTree` représentant un noeud de l'arbre, cette classe a donc 4 attributs de type `QuadTree` correspondant à ses 4 fils
   - Une classe `QuadLeaf` héritant de `QuadTree` représentant une feuille de l'arbre, chaque feuille contient au plus un `Point`
2. Implémentez un constructeur et une methode `toString` pour ces classe.

Dans la classe `QuadTree` :

3. Écrivez une méthode `getCenter` donnant le point au centre du carré formé par `(xmin, ymin)` et `(xmax, ymax)`
4. Programmez les fonctions `isInsideCircle` et `intersectsCircle` renvoyant un `true` si le `QuadTree` est totalement inclus (respectivement intersecte) un cercle étant donné son centre et son rayon.

III.2 Implémentation de la recherche

L'implémentation de la recherche se fera de manière récursive : une recherche dans un `QuadNode` revient à chercher dans chacuns de ses fils qui intersecte le cercle. Pour chercher dans un `QuadLeaf` il suffit de vérifier que son éventuel `Point` est à l'intérieur du cercle.

5. Ajoutez des méthodes abstraites `getPoints` qui permet de récupérer l'intégralité des points dans un `QuadTree` et `getPointsInCircle` permettant de récupérer les points d'un `QuadTree` inclus dans un cercle donné en argument par son centre et son rayon

Complétez la méthode `getPointsInCircle` de `QuadNode`

```
public LinkedList<Point> getPointsInCircle(Point center, double r)  {
  if (!this.intersectsCircle(center, r)) return new LinkedList<>();
  else if (this.isInsideCircle(center, r)) return this.getPoints();
  else {
    LinkedList<Point> res = new LinkedList<>();
    res.addAll(...);
    res.addAll(...);
    res.addAll(...);
    res.addAll(...);
    return res;
  }
}
```

6. Implémentez ces méthodes dans les classes `QuadNode` et `QuadTree` en suivant la méthode décrite précédemment

Bonus : Modifiez vos méthodes pour trouver le point le plus proche (sans passer par un tri de la sortie de `getPointsInCircle`)

III.2.1 Note

Nous avons ici implémenté un algorithme de recherche dans un arbre nommé DFS (Depth First Search), c'est une méthode de parcours d'arbre assez classique et simple à implémenter.

III.3 Construction du QuadTree

Maintenant que l'on peut rechercher dans un `QuadTree`, il ne reste plus qu'à le créer à partir d'une liste de points.

Pour plus de modularité, nous allons implémenter une méthode qui ajoute un point à un `QuadTree` déjà existant. Cette fonction aussi peut être récursive.

7. Créez et implémentez la méthode abstraite `insertPoint` prenant en argument un point et l'insérant dans l'arbre tout en conservant les contraintes de construction de l'arbre.
8. Programmez maintenant une méthode statique `fromPoints` utilisant la méthode de la question précédente pour construire un arbre à partir d'une liste de `Points`