Division des enseignements en informatique

TP Problème du plus proche voisin

Dans ce TP, nous nous intéressons au problème de la détermination du plus proche voisin dans un semi de points 2D. Cette problématique trouve de nombreuses applications (gestion administrative d'un territoire, interpolation spatiale, etc.).

Dans un premier temps, nous mettrons en place une méthode de résolution "naive" consistant à calculer la distance d'un point à tous les autres points pour ne retenir que la plus petite. Dans une seconde partie, nous mettrons en oeuvre une méthode d'indexation spatiale dans le but d'améliorer les performances de l'algorithme.

Tout au long de ce TP, un point sera modélisé par un tuple de 2 coordonnées réelles `(x, y)`.

Pour chaque fonction, prévoir la documentation et des tests unitaires lorsque cela s'y prête.

1 Les éléments communs

1. Ecrivez une fonction `distance(point1, point2)` retournant la distance entre les deux points en entrée de la fonction.
2. Ecrivez une fonction `point_aleatoire(xmin, ymin, xmax, ymax)` retournant un point dont les coordonnées x et y sont tirées aléatoirement dans l'emprise `((xmin, ymin), (xmax, ymax))`.
3. Ecrivez une fonction `points_aleatoires(n, xmin, ymin, xmax, ymax)` retournant une liste de n points tirés aléatoirement.

2 L'approche naïve

4. Ecrivez une fonction `plus_proche_voisin(p_ref, points)` prenant en entrée un semis de n points tirés aléatoirement ainsi qu'un point référence (x, y) et retournant l'indice du point du semi le plus proche du point référence ainsi que la distance les séparant.
5. Ne pas oublier la documentation et les tests unitaires.
6. En utilisant le module `time` et sa fonction `time()` prévoyez une fonction retournant les temps de calculs pour plusieurs tailles de semis de points.
7. Représentez ces temps de calculs dans un graphique `matplotlib`.
8. Indiquez dans la documentation de la fonction `plus_proche_voisin()` quelle est la complexité dans le pire des cas de l'algorithme utilisé. L'évolution des temps de calculs obtenus question précédente vous paraît-elle cohérente avec cette classe de complexité.

3 Calcul de plus proche voisin avec indexation spatiale

L'indexation est un mécanisme pour organiser des données et ainsi faciliter ultérieurement les recherches dans ces données [1]. L'indexation spatiale est une forme d'indexation utilisée pour optimiser les calculs impliquant des positions ou des distances. Il existe différentes formes d'index spatial (grille, quadtree, octree, etc. [2]).

Nous mettrons en oeuvre la plus "simple" des méthodes d'indexation spatiale qui consiste à regrouper les points par zones (dans notre cas les cases d'une grille). La recherche du plus proche voisin se limitera alors à la zone dans laquelle se situe le point de référence ainsi que les zones voisines.

10. Ecrivez une fonction `initialisation_index(points, xmin, ymin, xmax, ymax, nlig, ncol)` affectant chacun des points de la liste de points en entrée à une cellule de l'index. En sortie, chaque cellule de l'index doit contenir les indices des points qu'elle contient.

L'index est stockés sous forme de dictionnaire. Par exemple, si le semis se compose de 10 points aléatoires dont les coordonnées x et y sont comprises entre 0 et 10, et que l'on génère un index spatial de 2 lignes et 2 colonnes, le dictionnaire sera de la forme :

```
index = {
    'info': {
        'xmin': 0, 'ymin': 0, 'xmax': 10, 'ymax': 10,
        'dx': 5.0, 'dy': 5.0, 'nlig': 2, 'ncol': 2
    },
    0: {
        0: [2, 7],
        1: [1, 4, 9]
    },
    1: {
        0: [0, 3],
        1: [5, 6, 8]
    }
}
```

Soit : une première balise avec les informations générales sur l'index, puis une balise par ligne avec une balise par colonne et la liste des points que la cellule contient enfin.

Remarque : vous pouvez vérifier que pour un grand semis de le nombre de points dans chaque cellule tend à s'homogénéiser.

11. Ajoutez une fonction `indice_index(point, xmin, ymin, dx, dy)` retournant les indices de la cellule dans laquelle le point en entrée se situe.
12. Ecrivez une fonction `plus_proche_voisin_avec_index(point_ref, points, index)` retournant l'indice du point du semi le plus proche du point référence ainsi que la distance les séparant en utilisant l'index spatial en paramètre.
13. Ne pas oublier la documentation et les tests unitaires.
14. En utilisant le module `time` et sa fonction `time()` prévoyez une fonction retournant les temps de calculs pour plusieurs tailles de semis de points.
15. Représentez ces temps de calculs dans un graphique `matplotlib` pour différentes tailles de grille.

1. https://fr.wikipedia.org/wiki/Index_(base_de_données)
2. https://fr.wikipedia.org/wiki/Index_spatial

4 Conclusion

16. Comparez les performances des deux méthodes et indiquez dans quels cas la génération d'un index sera pertinente.