Algorithmes et Pensée Computationnelle

Spatial Algorithms

Le but de cette séance est de comprendre et d'implémenter des algorithmes spatiaux.

1 Nearest-Neighbor

Nous allons chercher à implémenter une recherche du plus proche voisin. Le but de cette méthode est de, étant donné un point de "départ" et un ensemble de point, trouver le voisin le plus proche du point de départ. Nous allons implémenter cette algorithme et ensuite l'étendre à un algorithme des k plus proches voisins (recherche des k voisins les plus proches plutôt que du seul voisin le plus proche).

Question 1: (*5 minutes*) **La fonction de distance : Python**

Pour implémenter notre recherche, nous avons besoin de coder une fonction permettant de calculer la distance entre 2 points. Programmez une fonction qui permet de calculer la distance entre 2 points.

Conseil

Soit 2 points en 2 dimensions (x_1, y_1) et (x_2, y_2), la distance euclidienne entre ces 2 points est donnée par : $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Solution

```
1  #Time to code !
2  import math #permet d'importer la library nécessaire au calcul de la racine carrée
3
4  def calculate_distance(point1,point2):
5      #Implémentez la formule de la distance euclidienne entre 2 point ici
6      return math.sqrt((point1[0]-point2[0])**2+(point1[1]-point2[1])**2)
```

Note : Vous auriez pu utiliser (...)**0.5 en lieu et place de la fonction math.sqrt(.)

Question 2: (*10 minutes*) **Nearest-neighbor search**

Implémentez la rechere du voisin le plus proche. Ce dernier fonctionne de la façon suivante :

1. Traversez chaque point.
2. Pour chaque point, calculez la distance entre le point et le point de départ.
3. Retournez les coordonnées du point le plus proche.

Conseil

Utilisez la fonction de distance de la question 1 et parcourez les points à l'aide d'une boucle for. Si votre input est [[2,3],[5,6],[1,4],[2,4],[3,5]] et que le point de départ est [4,4], alors l'ouptut devrait-être ([3,5] 1.414).

Note : Pour que votre programme fonctionne, programmez votre algorithme du plus proche voisin dans le même programme que celui de la Question 1.

Solution

```
#Question 2

def nearest_neighbor(start, point_set):#start correspond au point de départ, point_set correspond à l'ensemble
        des points
    for i in range(len(point_set)):#on parcourt tout les point de l'ensemble

        if i == 0:
            min_distance = calculate_distance(start, point_set[0])
            nearest_nei = point_set[0]
            #La distance minimale n'étant pas définie, on doit l'initialiser à la première itération, c'est ce qu'on fait ici

        distance = calculate_distance(start, point_set[i])

        if distance < min_distance:
            min_distance = calculate_distance(start, point_set[i])
            nearest_nei = point_set[i]

        #Cette partie du code détermine si le point actuellement considéré, est plus proche du point de départ que
         les points
        #parcouru jusqu'ici. Si c'est le cas, on redéfinit la distance minimale et "enregistre" les coordonées du point

    return nearest_nei, min_distance
```

Question 3: (*15 minutes*) **K-nearest-neighbor search**

Etendez l'algorithme du voisin le plus proche à un algorithme des K plus proches voisins.

Conseil

Appliquez l'algorithme du plus proche voisin K-fois. A la fin de chaque itération retirez le voisin le plus proche de l'ensemble des points sur lequel l'algorithme s'applique. De cette façon vous trouverez le second voisin le plus proche, le troisième, etc...

Votre fonction devrait retourner une liste de la forme : $[[x_1, y_2, distance1], [x_2, y_2, distance2], ...]$. Avec comme input [[2,3],[5,6],[1,4],[2,4],[3,5]], comme point de départ [4,4] et le nombre de voisins K=2, l'output de votre algorithme devrait-être : [[3, 5, 1.4142135623730951], [2, 4, 2.0]].

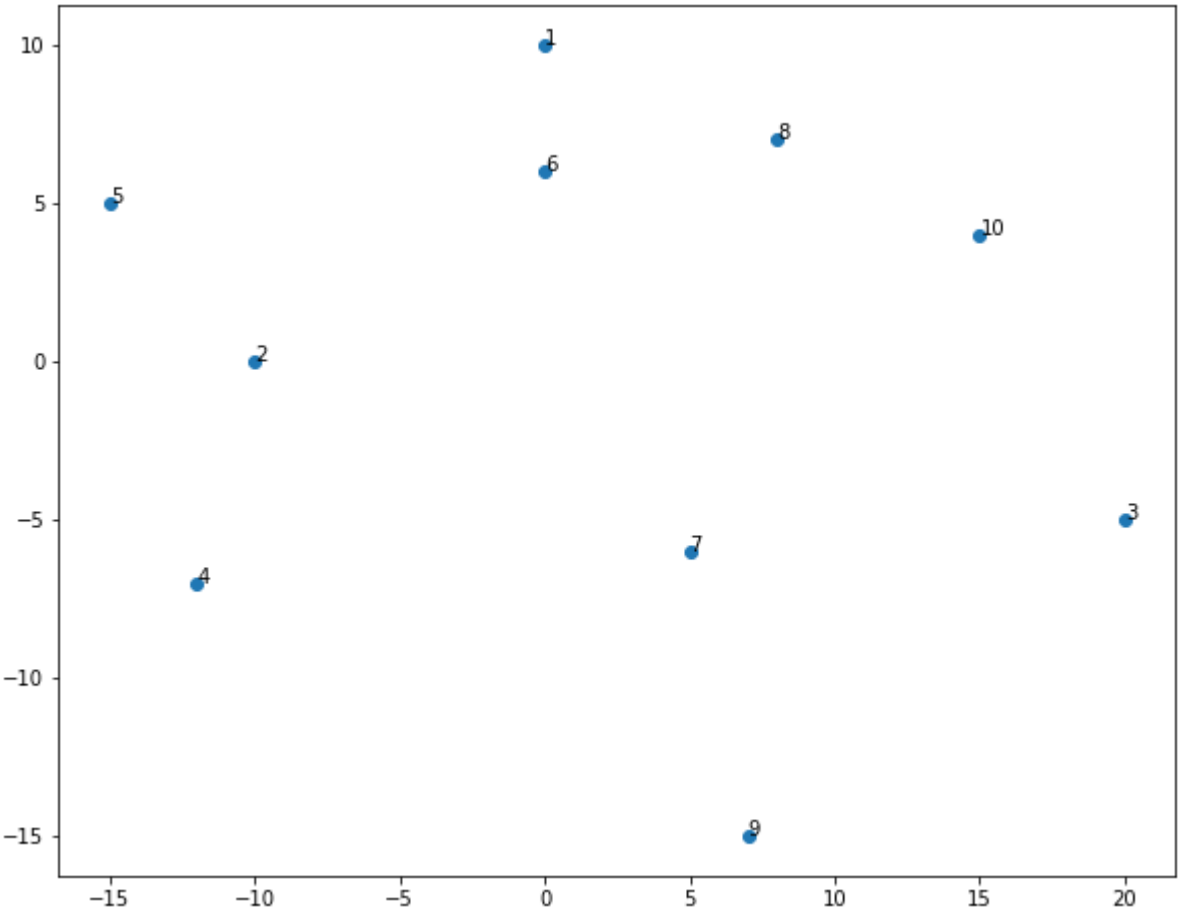
Note : Pour que votre programme fonctionne, programmez votre algorithme du plus proche voisin dans le même programme que celui de la Question 1 et la Question 2. Vous pouvez réutilisez le code des questions 1) et 2) pour cet exercice.

Solution

```
1   #Question 3
2
3   def K_nearest_neighbor(start,point_set, K):
4       temp = point_set #crée une copie de notre ensemble de point
5       print(temp)
6       k_nearest_nei = []
7
8       for j in range(K):#A chaque itération on applique l'algorithme du nearest neighbour mais sur un ensemble de
            point réduit
9           point, distance = nearest_neighbor(start,temp)
10          point.append(distance)
11          k_nearest_nei.append(point)
12          temp.remove(point)#On retire de l'ensemble de point le voisin le plus proche, de cette manière, à chaque
              itération,
13          #le voisin le plus proche sera de plus en plus éloigné.
14
15      return k_nearest_nei
```

2 K-dimensional tree

Question 4: (*5 minutes*) **KD-Tree, un échauffement : Papier**
Vous trouverez ci-dessous une liste de points numérotés de 1 à 10. Placez-les dans un KD-Tree et dessinez la séparation de l'espace qui en résulte.

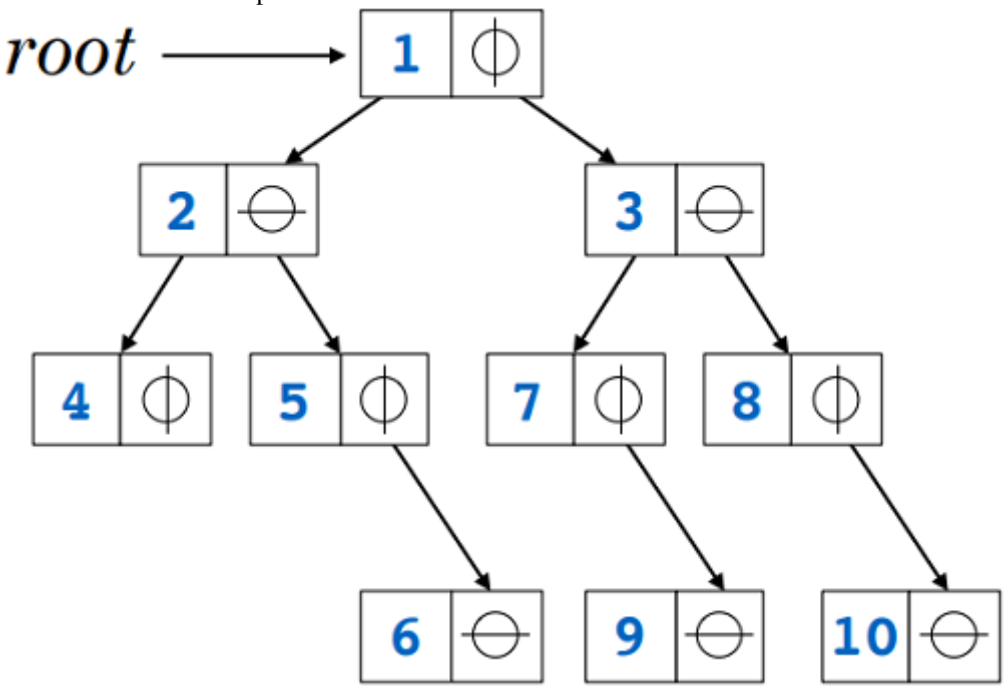


Conseil

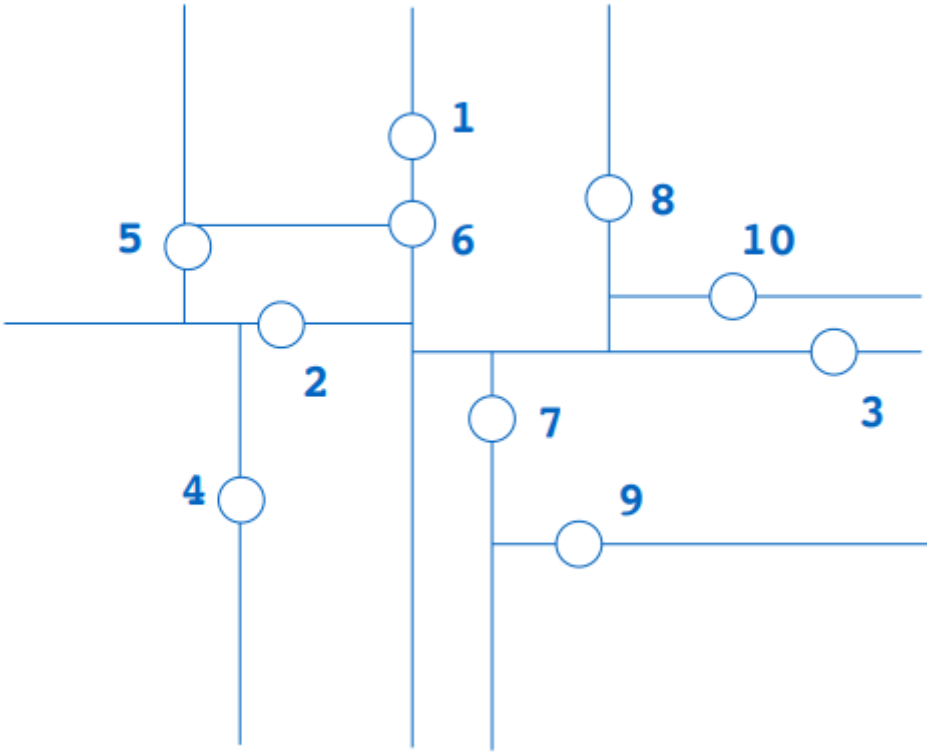
La première division se fait de façon verticale. Veillez à bien insérer les points dans l'ordre (point 1, point 2, etc..). Les noeuds se situant au même niveau devrait diviser l'espace selon le même axe.

Solution

Voici le KD-Tree correspondant :



Et la division de l'espace qui en résulte :



Question 5: (*15 minutes*) **KD-Tree : Python**

L'objectif de cette exercice est de programmer une fonction permettant d'ajouter un noeud à un KD-Tree. Les noeuds sont de la forme ((x,y), enfant à gauche, enfant à droite), x et y étant les coordonnées du noeud considéré. Chaque enfant peut être soit un noeud (même forme que expliqué précédemment) ou alors une feuille. Complétez le code contenu dans le fichier Question5.py.

Conseil

Veuillez vous référer aux pages 15 et 16 du cours de cette semaine. Ces pages expliquent le pseudo-code présent ci-dessous dont vous devez faire l'implémentation.
Voici le pseudo-code permettant d'ajouter un noeud à un KD-Tree :

```
ADD(node,point,cutaxis) :
        if node = NIL
                node ← Create-Node
                node.point = point
                return node
        if point[cutaxis] ≤ node.point[cutaxis]
                node.left = ADD(node.left, point, (cutaxis + 1) modulo k
        else
                node.right = ADD(node.right, point, (cutaxis+1) modulo k
        return node
```

Si votre réponse est correct, le code de Question5.py devrait print : [(0, 10), [(-10, 0), None, None], None].

Solution

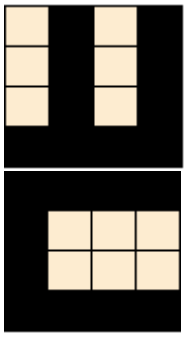
```
1  # Question 5
2
3  root = [(0,10), None, None] #Nous définissons ici juste la racine de l'arbre
4  k = 2 # Ici nous travaillerons en 2 dimensions
5  point = (−10,0) #Point que nous voulons ajouter dans le graphe
6
7  def add_node(node,point,cutaxis = 0):
8
9      if node is None: #Si le noeud n'existe pas, nous sommes donc dans une feuille, et il faut créer le noeud
10         node = [point,None,None]
11         return node
12
13     if point[cutaxis] <= node[0][cutaxis]: #1 Voir le pdf de correction.
14         node[1] = add_node(node[1], point, cutaxis + 1 % k)
15
16     else:
17         node[2] = add_node(node[2], point, cutaxis + 1 % k)
18
19     return node
20
21 add_node(root,point)
22
23 print(root)
```

Commentaire 1 : Si la coordonnée du point à ajouter est inférieure à celle du noeud selon l'axe de découpe en considération, alors le point doit se trouver dans le sous arbre de gauche. Par convention, le noeud de gauche correspond dans la liste [(x,y), noeud de gauche, noeud de droite] à l'indice 1, par conséquent, on appelle la fonction de façon récursive pour ajouter le point, mais cette fois-ci en partant de 1 cran plus bas dans l'arbre. Cela se répète jusqu'à ce qu'un ait atteint les feuilles et qu'un nouveau noeud doit être crée.

3 Quad-Tree

Question 6: (*10 minutes*) **Une mise en train : Papier**

Encodez les images ci-dessous dans un region Quad-Tree.



Conseil

Pour réussir cet exercice vous devez diviser chaque noeud en 4 sous-espaces de taille égale, et ce autant de fois que nécessaire pour que chaque zone blanche ou noir soit délimitée clairement. La branche la plus à gauche correspond au quadrant NW puis en allant de gauche à droite : NE, SE, SW.

Hint : Votre arbre devrait avoir une profondeur de 2 et disposer de 16 feuilles.

Solution

Image 1 :

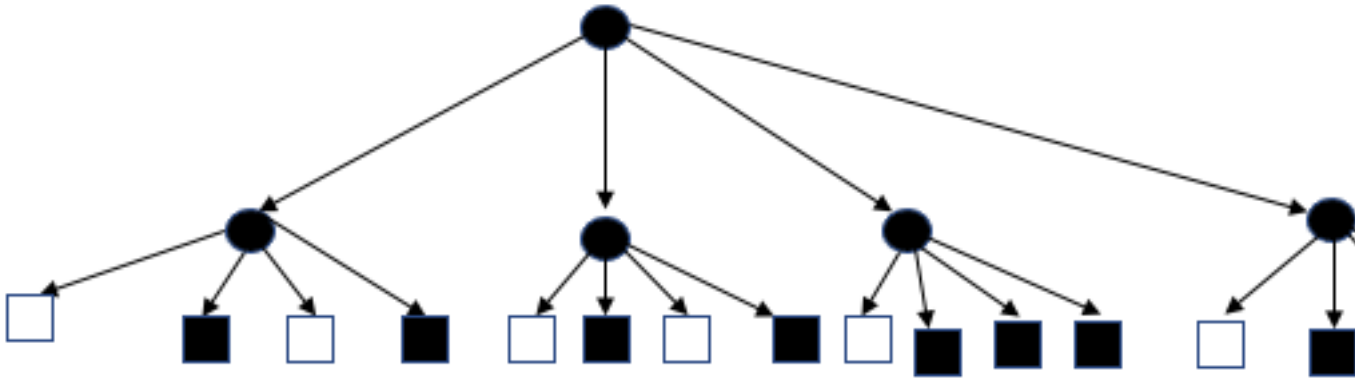
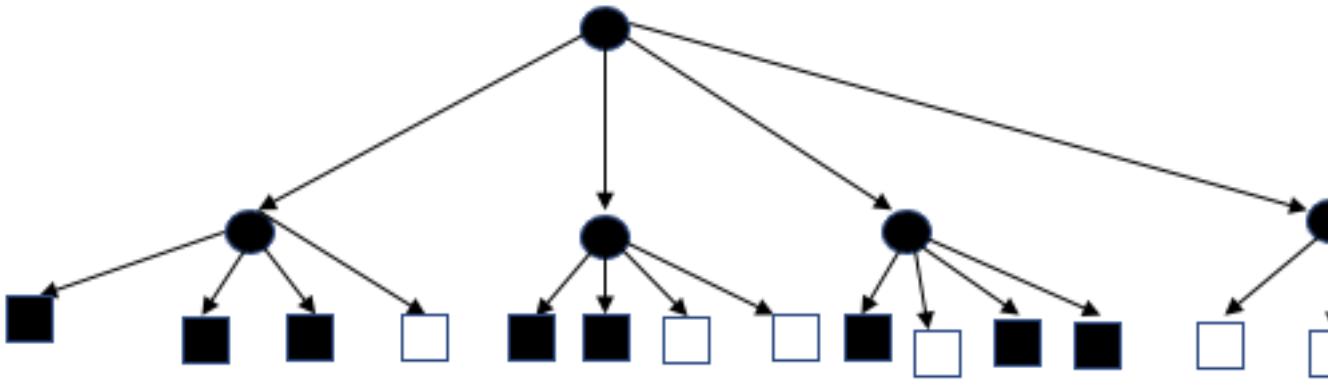
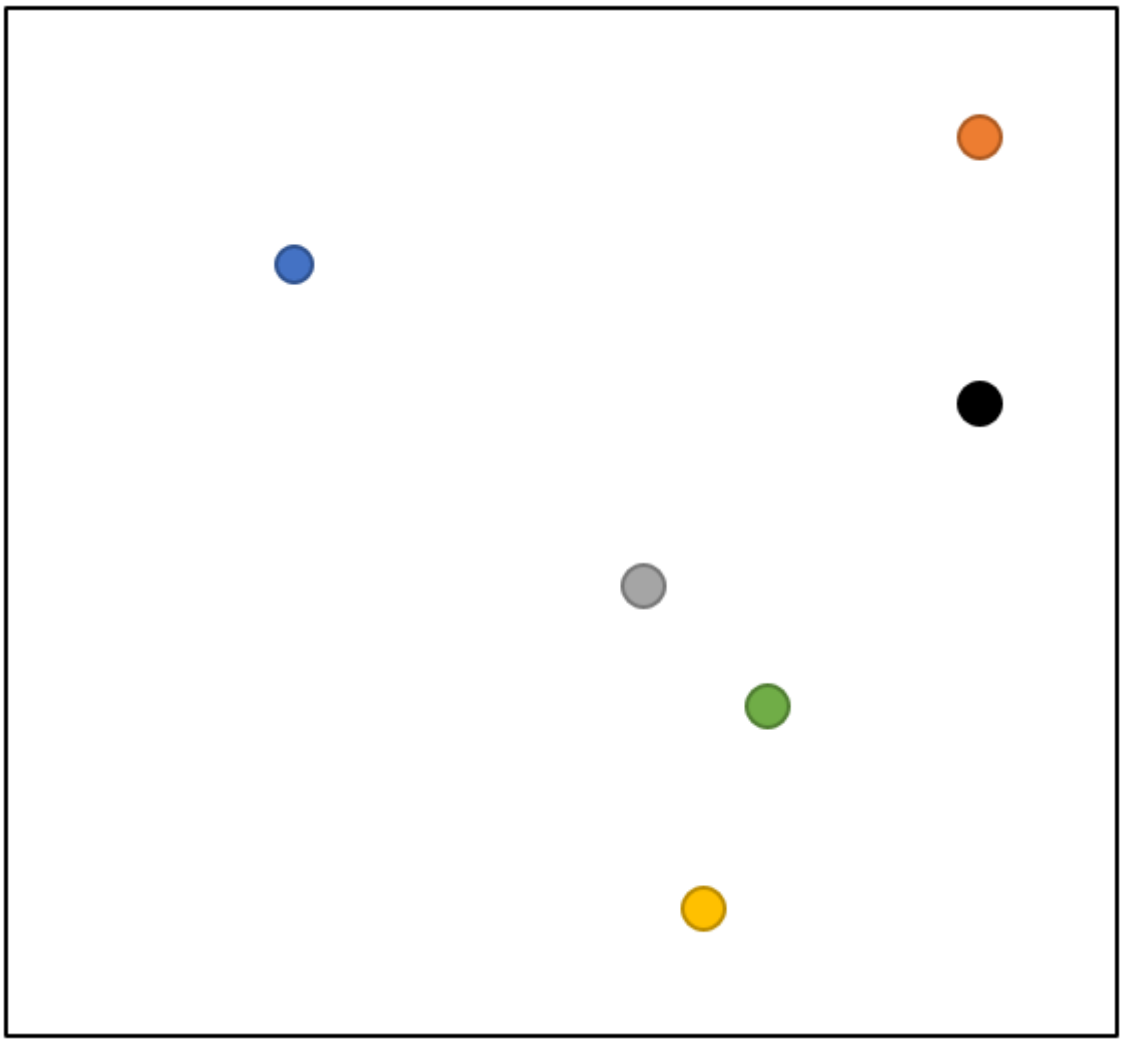


Image 2 :



Question 7: (*10 minutes*) **Une mission capitale : Papier**

Récemment embauché par la CIA, vous êtes à la recherche d'un individu se cachant dans une des villes suivant : Bleu, Orange, Noir, Gris, Vert et Jaune. Votre mission, si vous l'acceptez, est de créer un Quad-Tree qui vous permettera de géolocaliser le criminel de façon efficace. Vous trouverez ci-dessous une carte de villes. Créez le Quad-Tree et rétablissez la justice.



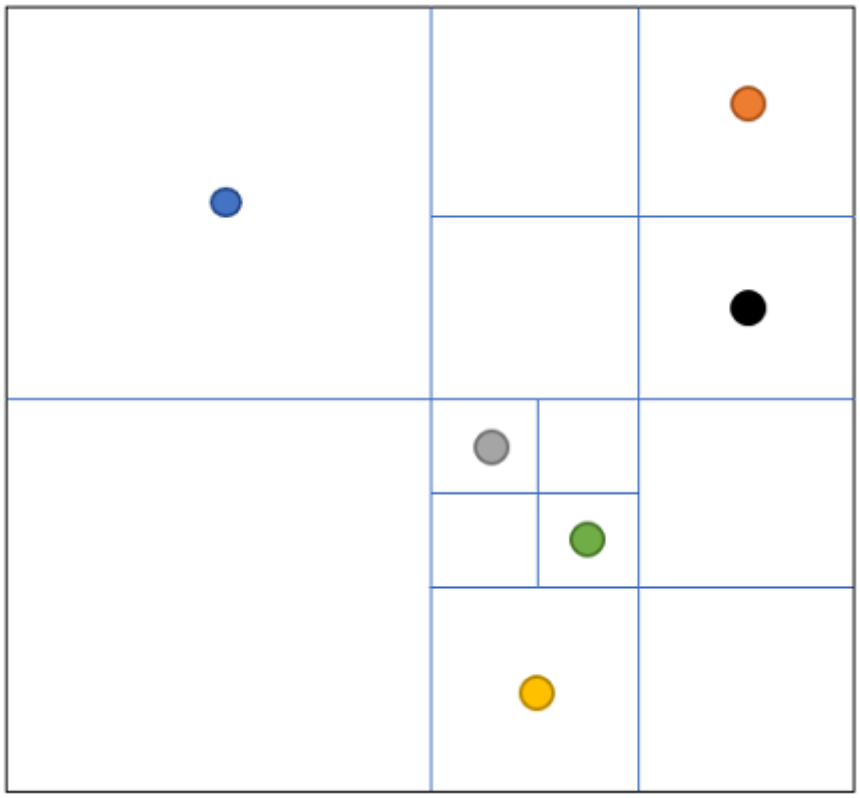
Conseil

Commencez par diviser la carte de la ville de la façon adéquate puis construisez le graphe.

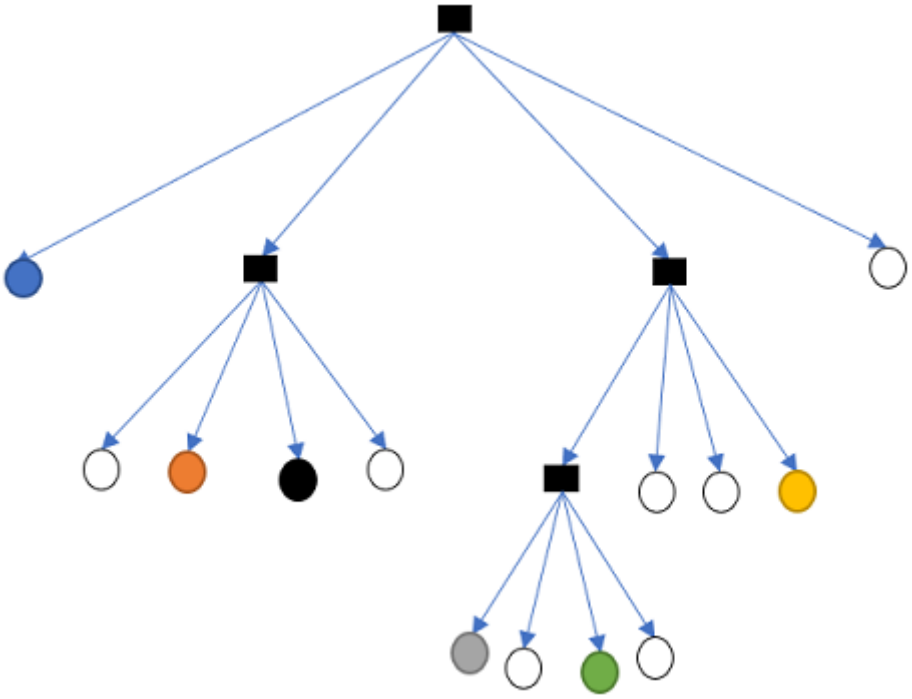
Hint : Les différentes branches de l'arbre n'auront pas toutes la même profondeur.

Solution

Voici la division de la carte qui permet de construire le Quad-Tree :



Le Quad-Tree qui en résulte :



Note : Un rond blanc correspond à un quadrant vide.