Information Systems - Lab assignment 3

Siemen Boelkens - s2788845
Hichem Bouakaz - s2525763

December 7, 2018

1 Implementing the QuadTree

To implement the Quadtree recurse function, we call the function with the initial BoundingBox, we calculate the centroid of the BoundingBox then we split the BoundingBox into four equal BoundingBoxes then we keep splitting recursively till we reach the depth of the QuadTree, see the provided code below:

```
1  def recurse(self,bbox, depth):
2    """
3    Internal function called on class contruction, this should
4    create the BoundingBoxes.
5
6    :param bbox: the initial BoundingBox
7    :param depth: the depth of the QuadTree
8    """
9    if depth >= self.depth:
10     return
11
12   centroid = bbox.centroid()
13   min_x = bbox.lower_left()[0]
14   min_y = bbox.lower_left()[1]
15   max_x = bbox.lower_left()[0] + bbox.width()
16   max_y = bbox.lower_left()[1] + bbox.height()
17
18   matrix = [
19     [min_x, min_y],
20     [max_x, min_y],
21     [min_x, max_y],
22     [max_x, max_y]
23   ]
24
25   for i in range(0,4):
26     box = bb.BoundingBox(centroid[0], matrix[i][0], centroid[1], matrix[i][1])
27     self.quads[depth].append(box)
28     self.recurse(box, depth + 1)
```

For test the Correctness of the code we run:

```
python3 plot_kdtree.py -quadtree 4 --quadshow True
```

We Obtained the Quadtree shown in Figure 1

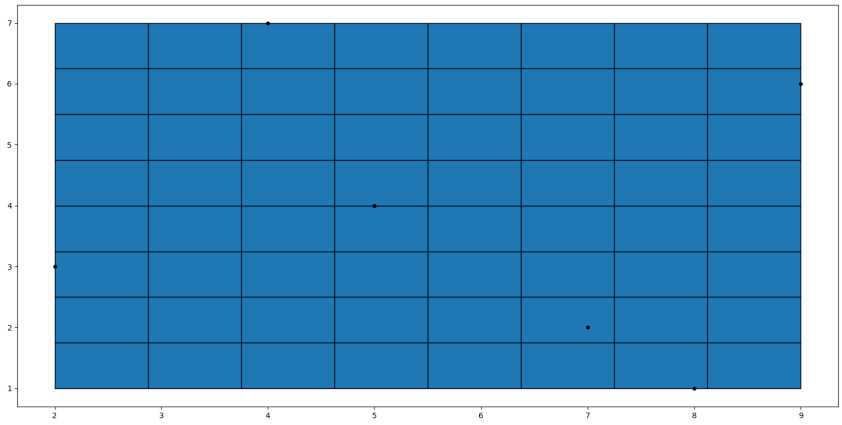

Figure 1: 8x8 chessboard pattern after implementing QuadTree.

2 Implementing the KDTree

To implement the KDTree we had to write the missing code in the function closest, we had to traverse through the KDTree and at each node we check if the axis to know on which axis the KDTree was split, then we compare the partition with the value of the point at the right axis to go left or right, then we call recursively the function till we reach the node that contains the leafs, and we return the all the leafs in the node. See the code below:

```
def closest(self, point, sidx = si.StorageIndex()):
  if "elements" in self.storage[sidx.storage()]:
    return self.storage[sidx.storage()]["elements"]

  axis = self.storage[sidx.storage()]["axis"]
  partition = self.storage[sidx.storage()]["partition"]

  x = point[0] if axis == 0 else point[1]

  if x <= partition:
    return self.closest(point, sidx.left())
  else:
    return self.closest(point, sidx.right())
```

3 Using the QuadTree depth to subsample the KDTree

To sample the KDTree using the Quadtree we have followed the given steps in the assignment:

- we did set up quad field for all records to the max quad-level in the database
- we iterate through all levels of the quadtree in reverse order.
- for every box inside that level we obtain its centroid.
- we use the centroid to obtain the list of closest points inside the KDTree
- we find the closest point inside that list
- finally update the quad field corresponding to that point to the current quadlevel

By implementing the steps only the closest points to the quad centroids that have level smaller than the sampling quadlevel will be plotted.
To check the implementation see the code below:

```
    if args.quadtree:

        quadlevel = args.quadlevel

        for record in dtb.query(dtb.keys()):
            dtb.update_field(record[0], 'quad', args.quadtree)

        quadtree = qt.QuadTree(tree.bounding_box(), args.quadtree)
        tree_quads = quadtree.quads

        # looping through the levels in reverse
        for idx in range(len(tree_quads) - 1, -1, -1):
            level_quads = tree_quads[idx]
            for quad in level_quads:
                centroid = quad.centroid()
                closest_records = dtb.query(tree.closest(centroid))
                min_dist = sys.maxsize
                # finding the closest point
                closest = closest_records[0]
                for close in closest_records[1:]:
                    distance = math.sqrt(((centroid[0] - close[1]) ** 2) + ((
centroid[1] - close[2]) ** 2))
                    if(distance< min_dist):
                        min_dist = distance
                        closest = close
                # updating the quad field of the closest point
                dtb.update_field(closest[0], 'quad', idx)
```

To compare our obtained results to the given results we had to run our code with the given parameters :

```
python plot_kdtree.py --filename ../data/addressen_groningen/adressen_groningen.shp
    --bbox-depth 8 --max-depth 9 --plot kdtree-bb --quadtree 4 --quadlevel 3
```

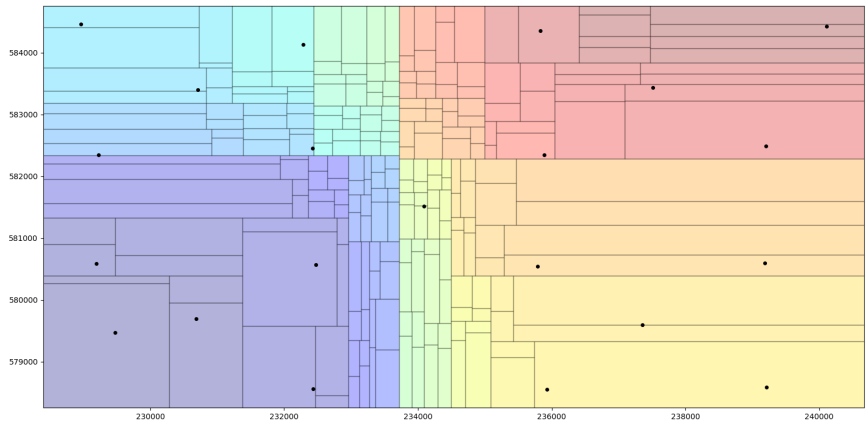

Figure 2: Sampling using Quadtree depth 4 and level 3.

```
python plot_kdtree.py --filename ../data/addressen_groningen/adressen_groningen.shp
    --bbox-depth 8 --max-depth 9 --plot kdtree-bb --quadtree 8 --quadlevel 7
```

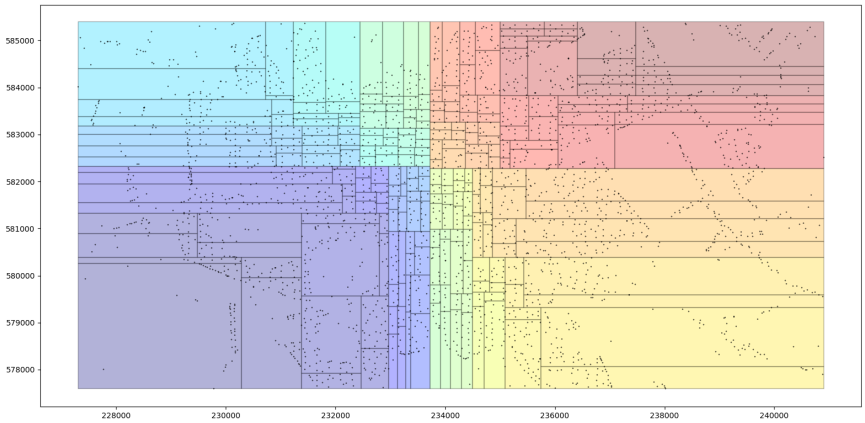

Figure 3: Sampling using QuadTree depth 8 and level 7.

Indeed the obtained Figures are identical to the given in the assignment.