Introduction to Spatial Computing- CSE 555

Homework 2

Due date: Sept 15, 2016 6:00am

Important Instructions:

- All submissions must be made through usebackpack site for this course (https://www.usebackpack.com/iiitd/m2016/cse5isc)
- Only one submission per team would be considered and graded. It would be assumed that all members of the team have participated equally and same score would be given to all members of the team.
- Your submission should have names of all the members of your team.
- Only one submission should be uploaded per team.
- Any assumptions made while solving the problem should be clearly stated in the solution.
- As always correctness of the algorithm must be ensured.
- TAs would be quizzing you on your code. You must understand each and every line of your submitted code. Also the implementation specifications mentioned in the questions need to be strictly followed. Failure to adhere to these requirements would result in substantial loss of points.
- Question 2 is for teams of size 3. This questions will not be graded for teams for size 2 or less.

Question 1 (80 points) (Programming assignment on spatial data):

In this question you would be evaluating KD tree and Region Qaudtree for range queries. To get started, first, create the following three synthetic datasets. Here, each data point is a (X,Y) coordinate. In each of these datasets, the points are distributed slightly differently.

Datasets for evaluations:

Following three synthetic datasets should be created and used for evaluation:

Dataset A: 1000 points generated in a uniformly random fashion where the x and y coordinates are integers between 0 and 70.

Dataset B*: 3*0 points generated in a uniformly random fashion where x and y coordinates are integers between 60 and 70. And another 970 points generated over the ranges of 0 – 60 (integer x and y coordinates)

Dataset C: 1000 points generated in a uniformly random fashion where x coordinate is an integer between 0-10 and y coordinate is an integer between 0 and 700.

Techniques to be implemented:

(a) KD tree
(b) Range query algorithm for KD tree (for rectangular ranges)
(c) Region quadtree
(d) Range query algorithm for region quadtree

Some Specifications for the KD-tree implementation to ensure uniformity across class

(1) Data structures for internal nodes, leaves, lines, and regions should be clearly defined. Internal nodes should store the following: (a) lines used to split, (b) data point used to split, (c) pointer to a left and right child, (d) if the current node is a left or a right child of the parent, (e) pointer to the parent.

(2) Points on the line go to the left child. Left child of an internal node is left of the line or below the line.

(3) To make things simple, at root level define region as the largest rectangle enclosing all the points in the dataset. This means that regions at the internal node level are all subsets of this grand region defined at the root.

(4) The tree must be built using a recursive function which carries the whole set of points (dividing them accordingly). Point by point insertion into KD tree is not allowed.

(5) At every level, you should choose the axis which has the largest spread and find the median data point by ordering the data on the chosen axis.

(6) Region corresponding to a node should not be stored in the internal node, rather it should be generated on the fly using the lines stored in the parents. You can also generate the region on-the-fly while traversing the path from root to the current internal node.

(7) Code must a have a function named "TestIntersect" which takes two regions and returns a Boolean "Yes" if they overlap.

(8) Code must a have a function named "PointInside" which takes a point and a region and returns a Boolean "Yes" if the point is inside the region.

(9) Code must a have a function named "Inside" which takes Region R1 and Region R2 and returns a Boolean "Yes" if R1 is inside R2.

(10) Code should also have function named "Visualize" which displays the KD tree by printing the internal node information level-wise. TAs would use this function to test your code for some sample input (max 15 data points) to see if the tree is being generated correctly.

(11) Your range query algorithm should return same answer as a naïve algorithm which goes through the entire dataset to return points which are inside the given query rectangle. Points on boundary are termed as being inside the rectangle.

Some Specifications for the Region Quadtree implementation to ensure uniformity across class

(1) Data structures for internal nodes, leaves, and regions should be clearly defined. Internal nodes should store the following: (b) data point used to split, (c) pointers to the children, (d) if the current node is a NW, NE, SE or SW child of the parent, (e) pointer to the parent.

(2) Points on the line go to the NW child.

(3) To make this simple, at root level define region as the largest rectangle enclosing all the points in the dataset. This means that regions at the internal node level are all subsets of this grand region defined at the root.

(4) The tree must be built using a recursive function. Point by point insertion into tree is not allowed.

(5) At level, you should split the given region into four equal parts. A region is not split further if has only one data point or no data point. In such a case you would make that as a leaf node.

(6) Region corresponding to a node should not be stored in the internal node, rather it should be generated on the fly using the information stored in the parents. You can also generate the region on-the-fly while traversing the path from root to the current internal node.

(7) Code must a have a function named "TestIntersect" which takes two regions and returns a Boolean "Yes" if they overlap.

(8) Code must a have a function named "PointInside" which takes a point and a region and returns a Boolean "Yes" if the point is inside the region.

(9) Code must a have a function named "Inside" which takes Region R1 and Region R2 and returns a Boolean "Yes" if R1 is inside R2.

(10) Code should also have function named "Visualize" which displays the tree by printing the internal node information level-wise. TAs would use this function to test your code for some sample input (max 15 data points) to see if the tree is being generated correctly.

(11) Your range query algorithm should return same answer as a naïve algorithm which goes through the entire dataset to return points which are inside the given query rectangle. Points on boundary are termed as being inside the rectangle.

Experiments to be conducted:

Experiment 1:

For each dataset, report the height of the tree for both KD tree and region quadtrees.

Experiment 2:

Dataset to be used: Dataset A

Algorithms to be compared: KD tree and Region Quadtrees

Variable Parameters: Area of the rectangle defining the range query. Vary the area as 50, 100, 150, 200, and 250.

Metric to be measured: Average (and standard deviation) fraction of number of internal nodes explored for 8 random queries of a particular area value. Note given an area, e.g., 50, you can create multiple rectangles of this area all over the dataset. Some of the rectangles should be long and thin and others more balanced.

Experiment 3:

Repeat experiment 2 with Dataset B.

Experiment 4:

Repeat experiment 2 with Dataset C.

Question 2 (40 points) (Programming assignment on spatial data):

For this question you need to implement a point quadtree. As you know that point quadtrees are extremely sensitive to order in which the points are inserted. As an attempt to address this limitation (to some extent), for each dataset, create 3 different random orderings of data points and construct a point quadtree on each one of them. For each dataset, choose the tree with least height amongst the three and discard the rest. Develop a range query algorithm for point quadtrees and repeat experiments mentioned in the Question 1.

Things to be submitted:

A zipped folder containing the following items. Please include your team information with the submission.

(a) Code, query processing algorithm (in a pdf), plots of Question 1. Also include a brief analysis of the trends obtained in the experiments (about 400 words). This analysis should contains two things: (1) individual level performance of algorithms implemented in Question 1 on dataset A, dataset B and dataset C; (2) relative comparison amongst algorithms implemented in Question 1 in terms of their performance on dataset A, dataset B and dataset C.

(b) Code, query processing algorithm (in a pdf) and plots for Question 2. Also include a brief analysis of the trends obtained in the experiments. This analysis should contains two things: (1) comparison between point quadtree and the algorithms implemented in Question 1 in terms of their performance on dataset A, dataset B and dataset C; (2) Individual performance of point quadtree on dataset A, dataset B and dataset C (about 250 words).