PG Software Lab - CSP 509 / CSP 609

Mini-project Specification Phase A

Due date: Sept 14, 2018 11:55pm

Total Weightage 18%

Important Instructions:

- All submissions must be made through the Moodle site for this course
- Any assumptions made while solving the problem should be clearly stated in the solution.
- Include your name and roll number in all the files.
- **Implementation specifications must be strictly followed. Failure to do so may lead to substantial loss of points.**
- **Very imp: There should be no directory structure in your code. All the files should be in just one directory.**
- As always correctness of the algorithm must be ensured.
- We may be quizzing you on your code. You must understand each and every line of your submitted code.

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

In this question you would be evaluating KD tree and Region Qaudtree for range queries. Following is the format for the input dataset. Here, Point ID, x-coordinate and y-coordinate are integers. Do not hard code any ranges for them as they can vary in the test datasets.

<Point ID> <x-coordinate> <y-coordinate> <new line>

…

…

…

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 root, internal nodes and leaves should be clearly defined. Internal nodes should store the following: (a) line used to split, (b) pointer to a left and right child, (c) if the current node is a left or a right child of the parent, (d) 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) At root level you can define region as the smallest 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. You can store the region (in form of xmin ymin xmax ymax) only at the root.

(4) The tree must be built using a recursive function which carries the whole set of points and divides 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 generate the region on-the-fly while traversing the path from root to the current internal node.

(7) At any stage during insertion, splitting should not continue further if the current region has only **alpha** #data points (parameter to be set in experiments) or less. In such a case, you need to create a leaf node. In this implementation, leaf node would just store a file name. This file (on your hard drive) would actually store all the alpha #data points. For each of these data points, store the Point ID, x-coordinate and y-coordinate in the file. Store each point in a new line.

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

(9) 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.

(10) 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.

(11) 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.

(12) 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. Your range query should use the functions defined in item (8), (9) and (10).

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

(1) Data structures for root, internal nodes and leaves be clearly defined.

(2) Internal nodes should store the following: (a) coordinate of the point used to split, (b) pointers to the children, (c) if the current node is a NW, NE, SE or SW child of the parent, (d) pointer to the parent.

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

(4) At root level you can define region as the smallest 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. You can store the region (in form of xmin,ymin, xmax, ymax) only at the root node.

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

(6) At any stage, you should split the given region into four equal parts. A region is not split further if has only **alpha** #data points (parameter to be set in experiments) or less. In such a case, you need to create a leaf node. In this implementation, leaf node would just store a file name. This file (on your hard drive) would actually store all the alpha #data points. For each of these data points, store the Point ID, x-coordinate and y-coordinate in the file. Store each point in a new line.

(7) 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 generate the region on-the-fly while traversing the path from root to the current internal node.

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

(9) 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.

(10) 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.

(11) 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.

(12) 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. Your range query should use the functions defined in item (8), (9) and (10).

Things to be submitted:

A zipped folder containing the following items. This zipped folder should be named as your “roll number.” Please include your name and roll number in all the files.

(a) Code for Region Quadtree and KD-tree
(b) Code for your naive algorithm
(c) Code for your range query algorithms