UG Software Lab - CS 409
Mini-project 2 Specification
Due date: 20-March-2021 11:55pm
Total Weightage 20%

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.
- 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.
- Very Important: Your code should not have a directory structure. All files (code + dataset + written material for questions) should be present in just one folder. Note that this is absolutely crucial for grading this assignment.

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

In this question you would be implementing the KD tree for k-nn 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>

…

…

Datasets for evaluations:

We would be using synthetic datasets for evaluation. Each data point in the dataset is a triple defined as follows: <id> <x-coordinate> <y-coordinate>. Here <id> is a unique number between 1 and number-points-in-dataset. Note that the spatial coordinates of the points in the dataset could be repeated. In other words, two different data-points (i.e., two different ids) can have the same x and y coordinates.

Dataset A: 30000 points generated in a uniformly random fashion where the x and y coordinates are random integers between 0 and 400.

Techniques to be implemented:

(a) KD tree
(b) K-nn query for the KD-tree
(c) A naive algorithm for finding the k-nn for a given query

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 the "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.. This leaf node would actually store all the alpha #data points. For each of these data points, store the Point ID, x-coordinate and y-coordinate. Store each point in a new line.

(8) Code should also have a 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.

(9) Your knn query algorithm should return the same answer as a naïve algorithm which goes through the entire dataset to return points which are closest to the given query point (ties broken arbitrarily).

Query Algorithm to be implemented on KD-trees:

Knn query: Given a query point q (x and y coordinate) and a value of k, retrieve the k closest points (from the dataset) for the input query point q. Correctness of the algorithm must be ensured. Grossly inefficient techniques and naive solutions would not be accepted.

Experiments:

Experiment 1

Algorithm: (a) Knn query algorithm on KD-tree
Alpha: 30
Dataset A
X-axis -- value of k (5, 20, 50, 100)
Y-axis --- Execution time.
Note: For a given value of k (e.g., 5), create several random query points inside the dataset boundary (i.e., between 0,0 and 400,400). Pass each of these as a knn query (with the given k value) and measure the running time; finally take an average of those values and plot it as the y-coordinate (x-coordinate is the value of k, 5 in this example).

Experiment 2

Algorithm: (a) Knn query algorithm on KD-tree
Alpha: 100
Dataset A
X-axis -- value of k (5, 20, 50, 100)
Y-axis --- Execution time.

Things to be submitted:

(a) All the code. Note that file names of code should have your roll numbers in the prefix. Code should not have a directory structure.

(b) Put the results of both the experiments in the same plot. Plot must be labeled properly (x axis, y axis and legends) and the legends must be clearly visible.

(c) Put all the plots in a short report and briefly comment on the observed trends. You should explain the observed trends appropriately.