# EC504

Nearest State and County

Gordon Wallace

Yang Hu

Haotian Cheng

**Introduction**

This project is to apply different algorithms in location searching practice. The purpose of implementations is to look for the nearest state and county for a given point within the United States. A large number of reference points from the official US Board on Geographic Names dataset are used to construct searchable data structures. A user provides a point with longitude and latitude and the program should search through data structures and provides 10 closest reference points. The closest 5 references points start a majority vote to determine which state and county the given point belongs to.

Three different algorithms, R tree, Quad tree and KD tree, are implemented to serve the same purpose as stated above. The details of each algorithm will be discussed. Also, the results and runtimes will be compared and analyzed further in the report.

Other than different algorithms for fundamental searching purpose, a Graphic User Interface (GUI) is also implemented to let user click on points on a map instead of entering coordinates manually, which is more convenient for users and testers to run the program. The user can choose from the provided algorithms to search for locations. A drop down list from 1 to 10 can be selected for the number of nearest reference point on display. Algorithm implementation and GUI establishment are all accomplished in Java under Eclipse environment.

**R Tree**

// description of implementation with emphasis on the design decisions related to data structures and algorithms

**Quad Tree**

To implement a Quadtree, we implement three important classes. Coord Class is used to store the coordinates with longitude and latitude. Node Class is used to store state, county and Coord correspond to the node point from our graph. Finally, we have QuadTree Class itself which can store its top left and bottom right coordinates which represent its border, QuadTree Children(NW, NE, SE, SW divided by its center) and Node if the QuadTree is a leaf.

With the three classes, the strategy to build a Quadtree data structure is very clear. First, we read the graph information and convert each point on the graph into a Node Class object. During the process, we find the graph border and we store all the Nodes in an ArrayList (java class). Then, we build the root QuadTree with the Graph border. Finally, we insert all stored Nodes in our list one at a time. Whenever a QuadTree contains more than one Node, it will divide into more Quadtrees as its children to store those Nodes.

In order to find Kth nearest points of a given point, we implement a search method in QuadTree class. Step 1, instantiate an Array with length K to store nearest points. Whenever a point is inserted into this array, the array get sorted by the distance to the given point simply using insertion sort. (because K is at max 10) Step 2, start from the root to see if it has child that may contain the target point(given point). Do recursive searching till we find the leaf cell that the target point cannot be any deeper. If the leaf cell contains any point, we store it to the array as one of the closest points for now. Step 3, loop through all the QuadTrees(cells) along the search path to see if the closest point it may contain is closer than the worst point stored in the array. If true, dig into that tree(cell) by repeating step 2. If it does have a closer point, replace the worst point in the array and sort the array.

**KD Tree**

KD tree is a binary search tree data structure that partitions the space to organize points. For a 2-D space, each coordinate divides the plane either horizontally or vertically. There are different ways of constructing a KD tree. The most common one is to sort all coordinates by their x values and take coordinate that has median x value, inserting as root of the tree. Now, the space is divided into halves. Performing sorting and finding median by the y values on each halve, the left side and right side of the root will be divided into halves again. Alternatively finding median of x and y values and divide the subspaces, the space will be divided into small rectangles. However, this method of inserting a KD tree requires sorting and finding median before inserting every node. Therefore, insertion will be relatively slow especially when there are a large number of coordinates. Another way to insert points is to insert without caring about finding medians. Picking an arbitrary point and divide the subspace where that point is located. The method without sorting might have small advantages on insertion, but it is also likely to construct a very unbalanced tree, which causes large runtime on searching.

In this project, the KD tree is implemented without sorting or finding median. Each node (defined as type N) in tree contains latitude, longitude, calculated x and y coordinates (stretch out the curve into a 2-D plane) and a left node and a right node. The tree contains the root node. To find the closest neighbors of a looked-up point, it goes through the tree and find the rectangle that contains the point, and sets the last node it has accessed as the current closest neighbor. Then starting from the root node again, going through the tree and look for closer neighbor that is even closer than the current closest. At each level of the tree, calculate either difference of x value between the point and the node or difference of y values between the point and the node . If the closest distance is smaller than the difference, the right side of that node isn’t worth looking into.

If only one closest neighbor is required, after each time finding a node that is even closer than the current closest, the current closest should get updated. However, in this case, 10 closest neighbors are required. Therefore, it is better to construct a list of nodes and keep updating that list (make the list sorted from smallest distance to largest). Keep the last node in the list as the “current closest”. It is very likely that within the circle of the looked-up point and current closest node, there are less than 10 nodes. Therefore, it is necessary to enlarge the range of searching by 5 km (it can be any number, but 5 km seems reasonable in this case) each time until 10 nodes are fully filled.

**Comparison**

Searching time of KD tree is extremely large since the tree itself isn’t constructed balancely. Although it saved some time on inserting without sorting and finding median, it also hindered the searching time as expected. In practice, insertion time may not be as important, since the coordinates are usually pre-loaded and users can access results more quickly if the tree is balanced.

**Project Features**

1. Storing points from NationalFile\_StateProvinceDecimalLatLong us using 3 different data structure (Quadtree, R tree, KD tree)

2. Searching Kth nearest point using 3 different algorithms (Quadtree searching, R tree searching, KD tree searching)

3. GUI for user to directly clicking various location on the US map and display its Kth nearest points and the state and county of the clicked point instead of entering coordinates manually.

**References**

SAMET, HANAN. “The Quadtree and Related Hierarchical Data Structures .” *UMIACS*, University of Maryland Institute for Advanced Computer Studies, 2 June 1984, <http://users.umiacs.umd.edu/~ramani/cmsc878R/p187-samet.pdf> .