COL106 : 2021-22 (Semester I)
Module 6: Places To Eat Near Me

Ashish R Nair and Chirag Bansal

October 15, 2021

1 Introduction

When ordering food online or deciding on a place to eat, you may often have searched 'places to eat near me' on Google or opened Zomato to look up for restaurants near you. In this module, you will implement a program that does exactly this – finds restaurants in the neighbourhood of users.

2 Problem Statement

You need to write a Java program to solve the 'places to eat near me' problem. Your Java program **must** solve this problem using 2-d trees **efficiently**. A description of the algorithms to build 2-d trees and count elements within a range has been provided in the next section.

The input to the program would be present in two files: `restaurants.txt` and `queries.txt`. `restaurants.txt` contains restaurant locations – latitude and longitude. `queries.txt` contains user query locations. The output file `output.txt` should contain the **number** of restaurants which are within ($\leq$) 100 latitude and 100 longitude units of users queries.

[Optional: Cost is an important deciding factor and a user may want to filter restaurants based on whether they are craving to eat some cheap fast food or whether they want to eat somewhere fancy. Implement a program that filters restaurants on three dimensions – latitude, longitude and cost. The user will provide a budget and the program should only find number of restaurants within the budget.]

3 2-d Tree Algorithms

1. To build a 2-d tree from a given collection of points follow this algorithm: At the root node, split the points based on the value of first dimension of the points. All the points whose value at the first dimension is less than or equal to the value of the median go to the left sub-tree and the rest go to the right sub-tree. For any node at depth d, if d is even, split the points based on the first dimension. Otherwise, split the points based on the second dimension. Continue this process until you reach the leaf nodes - nodes with one point. Also, at each node during this process, store the number of points under the

node and the orthogonal 'range' covered by the node. The range of a node is based on the splits from root to that node and is of the form $((x_{min}, x_{max}], (y_{min}, y_{max}])$, except when x_{max} or y_{max} is inf. ']' bracket indicates closed range with end point included whereas ')' bracket indicates open range. The range of the root is $((-inf, +inf), (-inf, +inf))$ and with each split, the range of the children can be found from that of the parent.

2. In the class, you looked at an algorithm to find/count points on one side of an orthogonal hyper-plane given a 2-d tree. Here, given a 2-d tree, we discuss a recursive algorithm to count the number of points within a rectangular range R. The algorithm is as follows: At the root node, find the intersection of R with the range of the children. Finding this intersection should be $O(1)$ using the range stored at each node. Consider a child c - if $range(c)$ is fully contained in R, then add the count of points in c to the total count of points within range R; if $range(c)$ is fully outside R then skip node c; else if $range(c)$ intersects R then recursively call the algorithm on c with c as the root. Repeat the same with the other child. On reaching a leaf node, simply see if the node is contained in R and update the count of points in range accordingly.

4 2-d Tree Solved Example

Consider the following collection of restaurants – { (1,2), (4,-6), (-2,-2), (-5,-7), (9,6), (5,5), (7,3), (2,8), (-1,4), (-8,-2), (-6,-3), (6,7), (8,-3), (-3,7), (-6,1), (-8,8) }, where for all points (x, y), x represents the first dimension or latitude and y represents the second dimension or longitude. The median of the points in the first dimension is -1 thus the root node will split the data based on whether the x coordinate is less than or greater than -1. *Note: The median of elements in case they are even in number can be taken as any of the two 'mid'-points. We are taking the left one.* We use the algorithm explained above to first split the data based on the first dimension, which gives us two sets -

- Left Sub-tree – (-2,-2), (-5,-7), (-1,4), (-8,-2), (-6,-3), (-3,7), (-6,1), (-8,8)
- Right Sub-tree – (1,2), (4,-6), (9,6), (5,5), (7,3), (2,8), (6,7), (8,-3)

Now, both of the sub-trees have depth 1 and will thus be split based on the y coordinates. Consider the left sub-tree co-ordinates, the median of the y coordinates of the points is -2 and the left sub-tree will further be divided into –

- Left-left Sub-tree – (-2,-2), (-5,-7), (-8,-2), (-6,-3)
- Left-right Sub-tree – (-1,4), (-3,7), (-6,1), (-8,8)

We keep applying this algorithm recursively until we get one point per leaf. The data in the tree contains two dimensions so we plot the data to understand the ranges represented by the 2-d tree in Figure 2. The red line represents the division at the root node – based on the x coordinate centered at -1. The left half and right half are then further divided based on the y coordinate as seen in the figure. The median of the y coordinates of the points in the left part is -2 thus the line $y = -2$ divides the left half. The right half is divided by the line $y = 3$. *Note: This is not the complete division of the 2-d tree.*

Based on the given points of data we create the tree in Figure 2. Here we represent each node by the line used to split the data, each node corresponds to a range of numbers and the

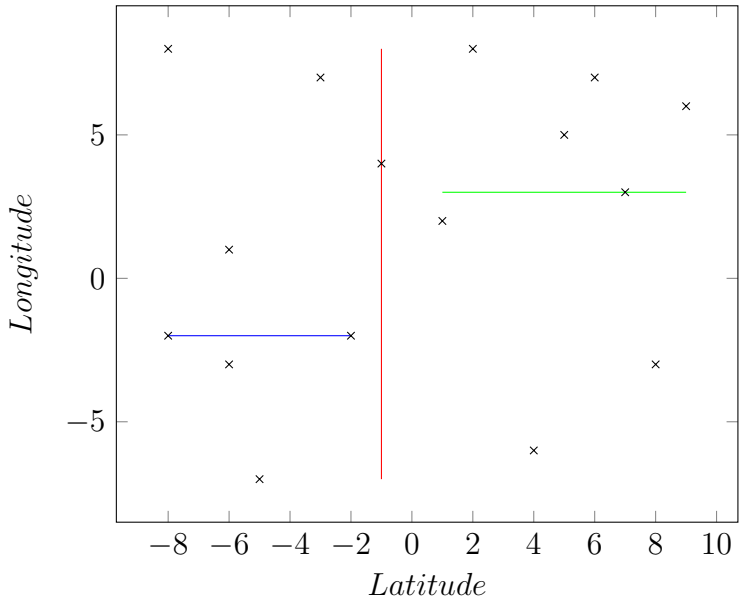

Figure 1: The collection of restaurants

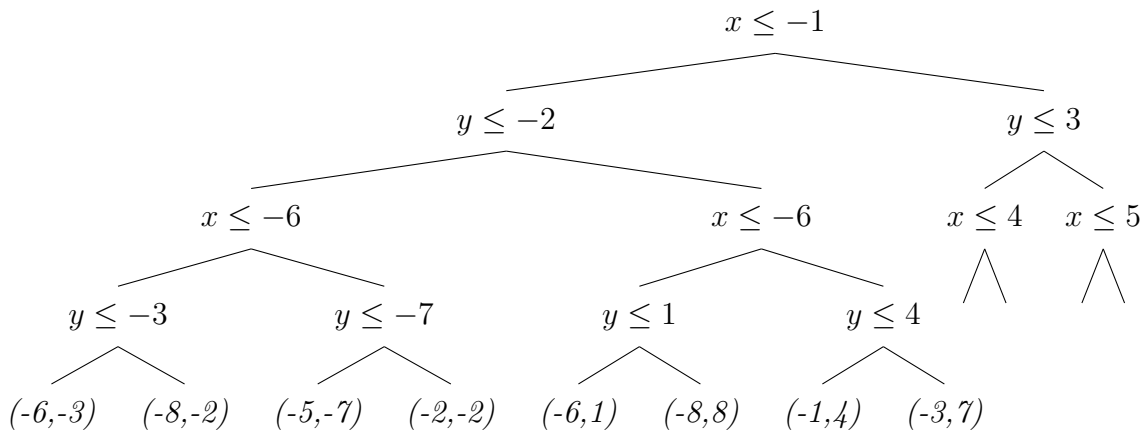

Figure 2: 2-d tree of given example

number of points under that node, for example the node $y \leq -2$, has 8 points in total and has range $(-inf, -1]$ for x coordinates and $(-inf, inf)$ for y coordinates. *Note: Since we are splitting the data based on the median, the tree will be balanced.*

Example of the range query: Let's say we wish to calculate the number of points (restaurants) contained in the range R = *[-6,-2],[1,5]*. At the root node we calculate the intersection of this range R with the ranges of the left and right sub-tree. As the right sub-tree has range $(-1, inf), (-inf, inf)$, the intersection is null and therefore we ignore this sub-tree. The range of the left sub-tree is not fully contained in R therefore we apply the algorithm recursively at that node. After calculating the intersection with ranges at this point, we see that the node to the left of $y \leq -2$ does not satisfy the range R. We keep applying the algorithm until we reach the node $(-6, 1)$. This is the only node that satisfies the given range and therefore the number of restaurants in this region is 1.
Note: The range includes the end points.

Sample I/O: The sample i/o provided has a list of restaurants and queries as per the spec-

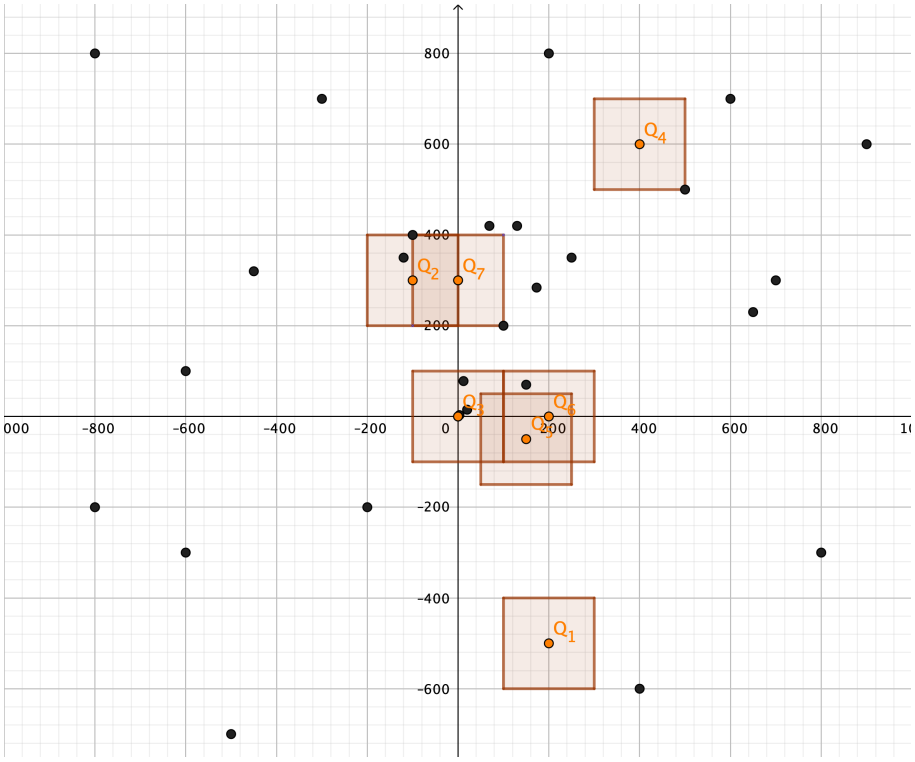

Figure 3: Representation of sample i/o queries

ifications given. Figure 3 displays the location of the restaurants and the squares representing the ranges of the queries.

Acknowledgement: Thanks to Aniruddha Deb for providing Figure 3 for the rest of the class.

5 File I/O Specifications

The contents of the files are as follows:

- `restaurants.txt` has one comma separated entry per line for each restaurant (the first line is column headings)

  latitude,longitude
  lat1,long1
  lat2,long2
  ... and so on

- `queries.txt` has one comma separated entry per line for each query (the first line is column headings)

  latitude,longitude
  lat1',long1'
  lat2',long2'
  ... and so on

- `output.txt` has one number in each line corresponding to the number of restaurants that satisfy each of the queries in `queries.txt`

  n1
  n2
  ... and so on

6 Submission Instructions

- Your filename must be `kdtree.java`. This file reads `restaurants.txt` and `queries.txt` and outputs the results in `output.txt`.
- You must create a directory whose name is your Kerberos id, followed by "Module6" (for example, if your Kerberos id is "xyz120100", then the folder name should be "xyz120100Module6"). The directory must contain only `kdtree.java`. Finally, compress this directory, and upload it on Moodle.

  The zipped file name should be `kerberosidModule6.zip` (that is, `xyz120100Module6.zip` in the above example).

7 Acknowledgements

This is Version 1 of the document. If you find any errors, please send an email to `kvenkata@iitd.ac.in`, `Ashish.R.Nair.cs517@cse.iitd.ac.in` and `cs5180402@iitd.ac.in` (and participate in the 'error-finding competition').
Many thanks to Soumil Aggarwal, Aradhye Agarwal, Hitesh Reddy, Vansh Kachhwal, Riya Sawhney and Dhruv Tyagi for finding errors in Version 0 of the lab-sheet.