95-771 Data Structures and Algorithms Project 2

Due: Tuesday, February 11, 2025, 11:59:59 PM

The objective of this assignment is to introduce the student to binary search trees and related algorithms by building a geometric application. This application will be based on 2d trees.

Our primary source of information on 2d trees is from an excellent video by Robert Sedgewick found at the following link. All that you need to know about 2d trees for this assignment is contained in this video.

https://www.youtube.com/watch?v=1OoM0phlO_U

The primary input to your application is a comma delimited text file containing crime records from the Pittsburgh area. These data were compiled during the 1990's (an exceptionally high crime decade in Pittsburgh). The file is named CrimeLatLonXY.csv and is found on the course schedule.

For each crime record, the file contains (X, Y) coordinate pairs in the state plane coordinate system. These (X, Y) coordinates are useful for calculating the distance between points (using the Pythagorean theorem). Each record also contains latitude and longitude coordinates. These coordinates are useful for displaying locations in GIS tools such as Google Earth. The other fields are not so important for our purposes. However, all of the data contained in the file will be represented in your tree.

The data structure that you will implement is a type of space-partitioning tree called a 2d Tree. We will use this data structure to store the crime records – you will store one crime record per node in the tree. Be sure to view the video mentioned above.

You will write several algorithms to manipulate the tree. These algorithms will be methods found in your TwoDTree class. The TwoDTree class will be found in a file name TwoDTree.java.

Required methods of the TwoDTree class:

1) public TwoDTree(String crimeDataLocation)

pre-condition: The String crimeDataLocation contains the path to a file formatted in the exact same way as CrimeLatLonXY.csv

post-condition: The 2d tree is constructed and may be printed or queried.

2) public void preOrderPrint()

pre-condition: The 2d tree has been constructed.

post-condition: The 2d tree is displayed with a pre-order traversal. Note: an example pre-order traversal appears on the course slides and will be discussed in class.

3) public void inOrderPrint()

pre-condition: The 2d tree has been constructed.

post-condition: The 2d tree is displayed with an in-order traversal. Note: an example in-order traversal appears on the course slides and will be discussed in class.

4) public void postOrderPrint()

pre-condition: The 2d tree has been constructed.

post-condition: The 2d tree is displayed with a post-order traversal. Note: an example post-order traversal appears on the course slides and will be discussed in class.

5) public void levelOrderPrint()

pre-condition: The 2d tree has been constructed.

post-condition: The 2d tree is displayed with a level-order traversal. Note: the level order traversal is not recursive. It uses a queue that you must write. This queue is defined in a separate file named Queue.java. Queue.java is built with a linked list based queue – using front and rear pointers. You will write

the methods to add to the rear and remove from the front of the queue. You will also design the signatures for these methods. An example level order traversal appears on the slides and will be discussed in class.

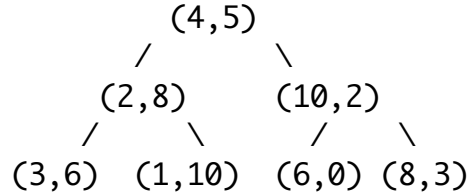
6) public void reverseLevelOrderPrint()

pre-condition: The 2d tree has been constructed.

post-condition: The 2d tree is displayed with a reverse level-order traversal.

Note: this reverse level order traversal is not recursive. It uses a queue that you must write. This queue is defined in a separate file named Queue.java. Queue.java is built with a linked list based queue – use the same queue that you built in question 5. It also uses a stack. This stack must be built on a linked list with only a front pointer. Name this class Stack.java.

Consider the following 2-d tree:



A reverse level order would visit the nodes in the following order: (8,3), (6,0), (1,10), (3,6), (10,2), (2,8), (4,5)

In your comments, provide a Big Theta value for your reverseLevelOrderPrint() and provide the detailed reasoning on how you reached that conclusion.

7) public ListOfCrimes findPointsInRange(double x1, double y1, double x2, double y2)

pre-condition: The 2d tree has been constructed

post-condition: A list of 0 or more crimes is returned. These crimes occurred within the rectangular range specified by the four parameters. The pair (x1, y1) is the left bottom of the rectangle. The pair (x2, y2) is the top right of the rectangle.

Note: The details involved in writing this method are found on the Sedgewick video. We will stay true to this video. In other words, alternative implementations are not allowed.

Note: The ListOfCrimes class must be written by you. It will be found in a file named ListOfCrimes.java. It will provide methods for adding crimes to the list and retrieving crimes from the list. It will be built using a singly linked list with a single head pointer. It will have a toString() method to return the list as a String and a toKML() method to return a KML representation of the list.

You will use the KML representation to view the crimes in Google Earth. An example KML file, containing a single crime, is shown here. You might want to load this text file into Google Earth to see how it is displayed. In your KML file, many crimes may be present. This is done by adding additional <Placemark> elements to the KML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<kml xmlns="http://earth.google.com/kml/2.2">
<Document>
  <Style id="style1">
  <IconStyle>
    <Icon>

<href>http://maps.gstatic.com/intl/en_ALL/mapfiles/ms/micons/blue
-dot.png</href>
    </Icon>
   </IconStyle>
   </Style>
   <Placemark>
     <name>ROBBERY</name>
     <description>5000 FORBES AV</description>
     <styleUrl>#style1</styleUrl>
     <Point>
       <coordinates>-
79.94295871,40.44471042,0.000000</coordinates>
      </Point>
    </Placemark>
</Document>
</kml>
```

8) public Neighbor nearestNeighbor(double x1, double y1)

pre-condition: the 2d tree has been constructed.
The (x1,y1) pair represents a point in space near Pittsburgh and in the state plane coordinate system.

post-condition: the distance in feet to the nearest node is returned in Neighbor. In addition, the Neighbor object contains a reference to the nearest neighbor in the tree.

Note: The details involved in writing this method are found on the Sedgewick video. We will stay true to this video. In other words, alternative implementations are not allowed.

Note: The Neighbor class must be written by you. It will be found in a file named Neighbor.java. Objects of class Neighbor will contain a distance field and a pointer into the 2-d tree.

Another class, TwoDTreeDriver is required. It will be contained within a separate file named TwoDTreeDriver.java. It will have a main method. When the main method is run it will load the crime data file into the 2d tree. It will interact with the user as shown below. You need not validate user input but you are required to fully implement each menu option. Only a subset of options is shown below. Output from the program is shown in blue. My comments on the execution are in parenthesis.

```
Java TwoDTreeDriver
Crime file loaded into 2d tree with 27218 records.
What would you like to do?
1: inorder
2: preorder
3: levelorder
4: postorder
5: reverseLevelOrder
6: search for points within rectangle
7: search for nearest neighbor
8: quit
>3 (This is a user selection from the menu. You may assume the
   user enters between 1 and 7. There is no need to check
   this value.)

(A level order list of crime data is displayed showing every
record in the file. It's too large to show here.)

What would you like to do?
1: inorder
2: preorder
3: levelorder
4: postorder
5: reverseLevelOrder
6: search for points within rectangle
7: search for nearest neighbor
8: quit
>6
Enter a rectangle bottom left (X1,Y1) and top right (X2, Y2) as
four doubles each separated by a space.

(Again, you may assume the user enters the data correctly.)

1357605.688 411838.5393 1358805.688 413038.5393

Searching for points within (1357605.688,411838.5393) and
(1358805.688,413038.5393)

Examined 25 nodes during search.
Found 9 crimes.

1358205.688,412438.5393,32898,5000 FORBES
AV,ROBBERY,1/25/90,140100,40.44471042,-79.94295871
```

```
1358701.856,412316.9622,32969,5050 FORBES
AV,ROBBERY,4/6/90,140100,40.44441059,-79.94116573
1358446.935,412903.5158,33181,1045 MOREWOOD
AV,RAPE,11/4/90,140100,40.44600282,-79.94213366
1358275.087,412559.1355,33347,1085 MOREWOOD
AV,RAPE,4/19/91,140100,40.44504608,-79.9427202
1358205.688,412438.5393,33570,5000 FORBES AV,AGGRAVATED
ASSAULT,11/28/91,140100,40.44471042,-79.94295871
1358205.688,412438.5393,34074,5000 FORBES
AV,ROBBERY,4/15/93,140100,40.44471042,-79.94295871
1358205.688,412438.5393,34590,5000 FORBES
AV,ROBBERY,9/13/94,140100,40.44471042,-79.94295871
1358349.449,412795.1595,33641,1060 MOREWOOD
AV,RAPE,2/7/92,140100,40.44569883,-79.94247415
1358646.638,412330.7924,35042,5044 FORBES
AV,ROBBERY,12/9/95,140100,40.44444479,-79.94136529

The crime data has been written to PGHCrimes.KML. It is viewable
in Google Earth Pro.

(The KML file will be named PGHCrimes.kml and will be viewable in
Google Earth Pro.)

What would you like to do?
1: inorder
2: preorder
3: levelorder
4: postorder
5: reverseLevelOrder
6: search for points within rectangle
7: search for nearest neighbor
8: quit
>7

Enter a point to find nearest crime. Separate with a space.
> 1359951.000 410726.000
(Again, you may assume the user enters the data correctly.)

Looked at 27 nodes in tree. Found the nearest crime at:
1359951.481,410726.1273,32874,320 SCHENLEY
RD,ROBBERY,1/1/90,140100,40.44013011,-79.93653583
```

```
What would you like to do?
1: inorder
2: preorder
3: levelorder
4: postorder
5: reverseLevelOrder
6: search for points within rectangle
7: search for nearest neighbor
8: quit

>8
Thank you for exploring Pittsburgh crimes in the 1990's.
```

Submission Guidelines

- Include comments in your code. You may reuse my pre- and post-conditions in your submission. If you write new methods, include pre- and post conditions of your own.
- If you feel that additional class files or methods are needed but they have not been specified here, by all means include them.
- A Google Earth viewable KML file should be included in your submission.
- A screenshot of Google Earth taken after a rectangle search is required. This screenshot will be a rendering of PGHCrimes.kml.
- Runtime analysis is only required on the levelOrderPrint and inOrderPrint methods.
- Place all project related files in a single directory and zip that directory. Submit one zipped directory to Canvas.