TEAM PROJECT 61

(Parth Bhatt,John Brugman,Vontrelle Collins, and Karan Dhawan)

SYSTEM DOCUMENTATION REPORT

PROJECT MILESTONE 4

ST_Contains

The function *ST_Contains* takes in a *queryRectangle*, which is a pair of Latitude and Longitude each separated by comma that construct a rectangle, as a string and a *pointString*, which is Latitude and Longitude for a single-point separated by comma, as a string. The goal of this function *ST_Contains* is to return true if the single-point,given as *pointString*, is on the boundary or inside the rectangle, given as *queryRectangle* else return false.

Step by Step explanation for *ST_Contains*:

1) A new variable *point* is created by removing the comma from the *pointString* using the *split* function and the string type of the *pointString* is converted to double using *map* and *iterator* function.
2) New variable *rectanglePoints* is created by removing the comma from the *queryRectangle* using the *split* function and the string type of the *queryRectangle* is converted to double using *map* and *iterator* function.
3) New variables *x* and *y* are created to separately store the latitude and longitude of the *point* using the *next* function.
4) New variables *x1* ,*y1*,*x2*,and *y2* are created to separately store the latitude and longitude of the *rectanglePoints* using the *next* function.
5) Now, the final step is to logically decide whether the data points *x* and *y* are in the rectangle created by the data points *x1*,*y1*,*x2*,and *y2*, including the boundary point, by using the if statement with condition (x >= x1 and x <= x2 and y >= y1 and y <= y2) and return true if the data point *x* and *y* is in the rectangle formed by *x1*,*y1*,*x2*,and *y2*.
6) If the condition mentioned in Step 5 returns false then the *if* statement will end and the function *ST_Contains* will return false.

ST_Within

The function *ST_Within* takes in *pointString1* and *pointString2*, which is Latitude and Longitude for a single point separated by comma, as a string and *distance* as double. The goal of this function is to return true if the distance between the two points is less than to the *distance* given else return false.

Step by Step explanation for *ST_Within*:

1. New variable *point1* is created by removing the comma from the *pointString1* using the *split* function and the string type of the *pointString1* is converted to double using *map* and *iterator* function.
2. New variables *x1* and *y1* are created to separately store the latitude and longitude of the *point1* using the *next* function.
3. New variable *point2* is created by removing the comma from the *pointString2* using the *split* function and the string type of the *pointString2* is converted to double using *map* and *iterator* function.
4. New variables *x2* and *y2* are created to separately store the latitude and longitude of the *point2* using the *next* function.
5. Euclidean distance *pointDistance*, the distance between two points in the plane with coordinates (x1, y1) and (x2, y2) is calculated using the Euclidean_Distance functions, that takes x1,y1,x2,and y2 as input parameters and returns distance according to the formula mentioned below:

   Euclidean distance((x1, y1), (x2, y2)) = SQRT((x1 - x2)² + (y1 - y2)²))

6. Now, the final step is to logically decide whether the *distance* given as input parameter is greater than or equal to the *pointDistance*, by using the if statement with condition (distance >= pointDistance) and return true if the *distance* is greater than or equal to *pointDistance*.
7. If the condition mentioned in Step 6 returns false then the *if* statement will end and the function *ST_Within* will return false.

PROJECT MILESTONE 5

HOTZONE ANALYSIS

For the Hotzone Analysis, there are two main files, *HotzoneAnalysis.scala* and *HotzoneUtils.scala*. The goal of these functions was to find the hotness of each rectangle by calculating the number of points within or on its boundary of that particular rectangle.

Step by Step explanation for object *HotzoneUtils*:

1. The object *HotzoneUtils* contains function *ST_Contains* that takes in a *queryRectangle*, which is a pair of Latitude and Longitude each separated by comma that construct a rectangle, as a string and a *pointString*, which is Latitude and Longitude for a single-point separated by comma, as a string. The goal of this function *ST_Contains* is to return true if the single-point,given as *pointString*, is on the boundary or inside the rectangle, given as *queryRectangle* else return false.
2. New Variable Array *x1*,*y1*,*x2*,and *y2* are extracted, which provides the latitude and longitude information of two points as type double forming a rectangle, from the *queryRectangle* using *split* and *map* functions.
3. *Math.min* and *Math.max* functions are used to find the new variables, min and max of the latitude and longitude (xmin, xmax, ymin ,ymax).
4. New Variable Array *x_point* and *y_point* are extracted, which provides the latitude and longitude information of a single point as type double, from the *pointString* using *split* and *map* functions.
5. Now, the final step is to logically decide whether the data points *x_point* and *y_point* are in the rectangle created by the data points *xmin,ymin,xmax,and ymax*, including the boundary point, by using the if statement with condition ((x_point >= xmin) && (x_point <= xmax) && (y_point >= ymin) && (y_point <= ymax)) and return true if the data point *x_point* and *y_point* is in the rectangle formed by *xmin,ymin,xmax,and ymax*.
6. If the condition mentioned in Step 5 returns false then the *if* statement will end and the function *ST_Contains* will return false.

Step by Step explanation for object *HotzoneAnalysis*:

1. The Object *HotzoneAnalysis* contains a function *runHotzoneAnalysis* which takes in a *pointPath*, *rectanglePath*, and a *spark Session*. The goal of this function is to run the Hotzone Analysis and output a number of points which are in each rectangle.
2. A new variable *pointDf* is created which reads the *pointPath* with a delimiter ";" creating a tempView *point*.

3. A new variable *rectangleDf* is created which reads the *rectanglePath* with a delimiter "/t" creating a tempView *point*.
4. Next, we join the two datasets while applying the function *ST_Contains* to see if the points are within the rectangle, and return the results in tempView *joinResult*.
5. Finally, we create a final view *orderedJoin*, where we query the previous table and select the rectangle, counts of points which are in the "hot zone" (as calculated in HotzoneUtils) and groups by rectangle, and ordered by rectangle. This final view *orderedJoin* would be returned as our final output for this section.

HOTCELL ANALYSIS

For the Hotcell Analysis, there are two main files, *HotcellAnalysis.scala* and *HotcellUtils.scala*. The goal of these functions was to return rectangles/cells based on their G score, which uses time and spatial relationships.

Step by Step explanation for object *HotcellUtils*:

The object *HotcellUtils* contains functions *inRectangle*, *nearby*, and *countNeighbor*.

A. Function *inRectangle*:
   The function *inRectangle* takes a *queryRectangle*, type String, and *pointString*, type String, as input parameters and returns true if the point is inside or on the boundary of the rectangle else false.

B. Function nearby:
   Function nearby takes a *pointString1*, type String, and *pointString2*, type String, as input parameters and returns false if the decimal degree (DD) difference between the latitude or longitude or timestamp of those *pointString1* and *pointString2* is greater than 1 else true.

C. Function *countNeighbor*:
   Function *countNeighbor* takes a *queryRectangle*, type String, and cell, type String, as input parameters and returns 27 or 18 or 12 or 8 depending on whether the cell's location is inside, face, edge or corner of the space time cube.

Step by Step explanation for object *HotcellAnalysis*:

1. The Object *HotcellAnalysis* contains a function *runHotcellAnalysis* which takes in a *pointPath* and a *spark Session*. The goal of this function is to run the Hotcell Analysis and output the cells based on their G score.

2. A new variable *pickupInfo* is created which loads the original data from the data source “*com.databricks.spark.csv*”.
3. Next, cell coordinates are assigned based on the pickup points.
4. Min and Max of each cell coordinate is calculated using the *coordinateStep* function defined in *HotcellUtils*.
5. New variables *numCells*, *rectangleZone*, and *weight* are derived from the min and max values of the cell coordinates.
6. We have imported *spark.implicits._* which supports implicit conversions.
7. In this step, Variable *pickupInfo* is updated with two columns point and cell with each data separated by “,” and tempView *pickupInfo* is also created.
8. Using function *inRectangle* defined in *HotcellUtils*, the number of points in the rectangle are returned using a query “select * from *pickupInfo* where *inRectangle('"+RectangleZone+"', point)*” and tempView *pickupInfo* is replaced.
9. Using query "*select cell, count(*) as count from pickupInfo GROUP BY cell*", the count of points in each cell are calculated.
10. New columns *squareCount* and *WX* are added to the tempView *pickupInfo*.
11. New variables *x_sum*, *x_sum_squared* and *avg_x* are defined to be used to calculate the G score.
12. Functions *nearby* and *countNeighbor* defined in *HotcellUtils* are used to replace the tempView *pickupInfo* with the cell, sum of the count*weight as *SUM_WX*, and countNeighbor as *COUNT_WX* by using cross join between two pickupInfo p1 and p2 where p1.cell is nearby p2.cell.
13. The new variable *getS* is defined to calculate the G score of each cell.
14. The new variable *DF* is created to include x,y,z,and score as the columns.
15. Dataframe *DF* is returned after the cells are ordered based on the query “*select x,y,z from DF ORDER BY score DESC, x DESC, y ASC, z DESC*”.

TEAM MEMBER PARTICIPATION

1. Initial team building call occurred to understand the background and expertise of each team member.
2. Before each project milestone due, discussion was held on Slack for the allocation of different tasks between the team members.
3. The task for each milestone was completed on time by each member.
4. During project milestone 5, a confusion regarding the task Hotzone analysis occurred and was resolved by communicating between team members over Slack Channel.