Juliana Porto

February 28, 2024

DS503 Big Data Management

Project 2

Question	Status	Comment
Q1	Fully Working	**<u>Step 1: Creating Datasets</u>** I created 2 datasets that are each over 100MB as outlined in the Project 2 Q1 Part1 instructions with the points in Points.txt and the rectangles in Rectangles.txt. I also created 2 smaller datasets with 10 points and 10 rectangles for testing purposes. *Assumptions:* • All values are integers • No rectangles can extend past the boundaries **Source Code:** Project2/Q1_SpatialJoin/Step1_CreatingDatasets/ Main.java **Points Dataset:** Project2/Q1_SpatialJoin/ Step1_CreatingDatasets/ Points.txt **Rectangles Dataset:** Project2/Q1_SpatialJoin/ Step1_CreatingDatasets/ Rectangles.txt **Test Datasets:** …/ Test_Points.txt …/ Test_Rectangles.txt
		<u>Step 2: Spatial Join</u> *Plan:* Point Map Logic: 1. For a given record, do parsing and extract fields 2. Single output (one for each point): a. Key = x value, y value b. Value = “Point”, x value, y value Rectangle Map Logic: 1. For a given record, do parsing and extract fields 2. Get max x & y of rectangle: a. max_x = bottomLeft_x + width

		b. max_y = bottomLeft_y + height 3. Get all possible points in rectangle from knowing min & max x & y 4. Multiple outputs (one for each point in rectangle)
		a. Key = x value, y value b. Value = "Rectangle", bottomLeft_x, bottomLeft_y, height, width Reduce Logic: 1. Separate array based on dataset source, either "Point" or "Rectangle" 2. Join records by x and y values 3. Print points within window a. If window is defined as an argument, use provided parameters b. Otherwise, use default values for entire grid: 0, 0, 10,000, 10,000 4. Output: c. key = NULL; d. value = bottomLeft_x, bottomLeft_y, height_ width, (x, y) **Source Code:** Project2/Q1_ SpatialJoin/Step2_SpatialJoin/SpatialJoin.java **Output with Small Test Dataset & no W():** Project2/Q1_SpatialJoin/Step2_SpatialJoin/ TestDataOutputQ1/part-r-00000 **Output with Small Test Dataset & W(3,4,7,8):** Project2/Q1_SpatialJoin/Step2_SpatialJoin/ TestDataOutputQ1_window/part-r-00000
		Additional Notes: • Single map-reduce job implemented for the spatial join operation (no points lost). • Optional input parameter W(x1, y1, x2, y2) that indicates a spatial window (rectangle) of interest within which we want to report the joined objects. If W is given, then any rectangle that is entirely outside W and any point that is outside W should be skipped. If W is omitted, then the entire two sets should be joined.

<table>
<tr><td></td><td></td><td>
Note: I needed to put W(x1, y1, x2, y2) in quotes when running in the terminal so that it didn’t think the parenthesis were special characters.
</td></tr>
<tr><td>Q2</td><td>Fully Working</td><td><u>Step 1: Dataset</u>
Used the point dataset created in Q1. The initial points in the HDFS file are totally in random order, and there is no specific organization. I also created 2 smaller datasets with a small number of points in specific areas for testing purposes.</td></tr>
<tr><td rowspan="2"></td><td rowspan="2"></td><td>Assumptions:
• All values are integers
Points Dataset:
Project2/Q2_OutlierDetection/ Step1_CreatingDataset/ Points.txt
Test Point Datasets:
.../Test_OutlierPoints.txt
.../Test_OutlierPoints0.txt</td></tr>
<tr><td><u>Step 2: Outlier</u>
Plan:
Point Map Logic:
1. For a given record, do parsing and extract fields
2. Assign point a current grid cell within the space
a. Make sure to round up if not an int
3. Assign point’s neighboring grid cells
a. Cell above
b. Cell below
c. Cell left
d. Cell right
e. Cell above left
f. Cell above right
g. Cell below right
h. Cell below left
4. If any grid cells are equal to 0 or above the max x & y value / r then do not print; otherwise…
5. Multiple outputs (one for each defined grid cell per point):
c. Key = grid cell
d. Value = x value, y value
Reduce Logic:</td></tr>
</table>

		1. Input: all points connected to a certain grid cell a. Gives us all points in that grid and its neighboring grids 2. For each point, count number of neighbors within radius r 3. If number of neighbors are less than k, print as outlier 4. Track non-outliers to identify unique outliers 5. Output: a. key = NULL b. value = x, y **Source Code:** Project2/Q2_OutlierDetection/Step2_ReportingOutliers/OutlierDetection.java **Test Outputs with Small Datasets:** .../TestDataOutputQ2/part-r-00000 .../TestData0OutputQ2/part-r-00000
		Additional Notes: • Single map-reduce job implemented for the outlier detection operation (no points lost). • Program takes in two mandatory parameters r and k. • Mapper has *max* variable to define x- and y-axis maximums and handle boundary restrictions. • Source referenced to understand partitioning and supporting (AKA neighboring) areas: https://web.cs.wpi.edu/~meltabakh/Publications/DOD-ICDE-2017.pdf
Q3	Partially Working	**Step 1: Dataset** Used the point dataset created in Q1. The initial points in the HDFS file are totally in random order, and there is no specific organization. Within the program I also created a list of centroid points that contains K initial seed points. I tried to make this save to HDFS but was unsuccessful. *Assumptions:* • Initial centroid values are integers, but then become doubles with averaging **Points Dataset:** Project2/Q3_Clustering/Step1_CreatingDataset/Points.txt

		Test Dataset: .../ Test0_Points.txt
		<u>Step 2: Clustering</u> *Plan:* Map Logic: 1. Generate all centroids, do parsing and extract fields 2. For a given point record, do parsing and extract fields 3. Assign each point to the closest center (k) a. $D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 4. Single output (one for each point): a. Key = closest centroid x, closest centroid y b. Value = x value, y value Reduce Logic: 1. Recompute all centroids a. x=average of all x-values b. y=average of all y-values 2. Output: a. key = NULL
		b. value = new centroid x, new centroid y Iterative Check: 1. Check if iterations have met max (6) a. If yes, then stop 2. If not at max, check if new centroids are the same as the previous a. If same, then stop 3. If not same, run another iteration to get new centroids and do everything again **Source Code:** Project2/Q3_ Clusering/Step2_Clustering/Clustering.java **Output with Small Test Dataset & k=3:** Project2/Q3_Clustering/Step2_Clustering/TestOutputQ3_k3/ iteration_0/part-r-00000 .../ iteration_1/part-r-00000 .../ iteration_2/part-r-00000 .../ iteration_3/part-r-00000 .../ iteration_4/part-r-00000 .../ iteration_5/part-r-00000

		Additional Notes: • Logic for stopping if new and old centroids the same not implemented