CSE 512 Course Project Operation Requirements

1. Operation Checklist

1) Geometry union
2) Geometry convex hull
3) Geometry farthest pair
4) Geometry closest pair
5) Spatial range query
6) Spatial join query

2. Operation Requirement

1) Geometry union

Definition: The union of a set S of polygons is the set of all points that lie in at least one of the polygons in S, where only the perimeter of all points is kept while inner segments are removed.

Example: The figures below show the input and output of an union operation. The input is a set of polygons and the output is the perimeter of the area which is composed of these polygons.

Function Name: *GeometryUnion*

Arguments:

(1) *String InputLocation*: the location of the input in HDFS
(2) *String OutputLocation*: the location of the output in HDFS

Requirement:

Load a set of polygons, output the union result of this set.

Input Dataset Schema:

x1, y1, x2, y2
// Every row is a pair of points (longitude, latitude) which defines a rectangle. This dataset has a bunch of rectangle.

Output Dataset Schema:

x, y
// Every row is a point (longitude, latitude). This dataset has a bunch of points. The result polygon is composed of all these points.

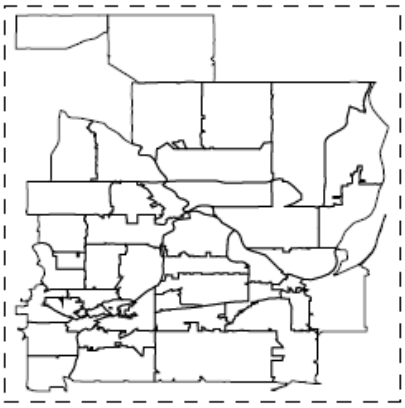

Figure 1 Union Input Polygons

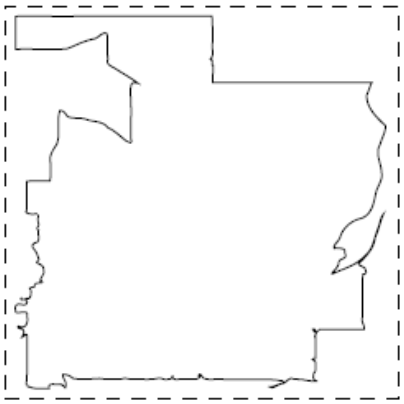

Figure 2 Union Result

Note: If the input data have rectangles that doesn't interact with each other, you should output one polygon that convers all rectangles.

2) Geometry convex hull

Definition: The convex hull of a set of points P is the smallest convex polygon that contains all points in P. The output of the convex hull operation is the points forming the convex hull ordered in a clockwise direction.

Example: The figures below show the input and output of a convex hull operation. The input is a set of points and the output is the points which compose the convex hull.

Function Name: *GeometryConvexHull*

Arguments:

(1) *String InputLocation*: the location of the input in HDFS

(2) *String OutputLocation*: the location of the output in HDFS

Requirement:

Load a set of points, output the convex hull of this set.

Input Dataset Schema:

x, y

// Every row is a point (longitude, latitude). This dataset has a bunch of points.

Output Dataset Schema:

x, y

// Every row is a point (longitude, latitude). This dataset has a bunch of points. A convex hull is composed of all these points.

Figure 3 Convex Hull Input Points

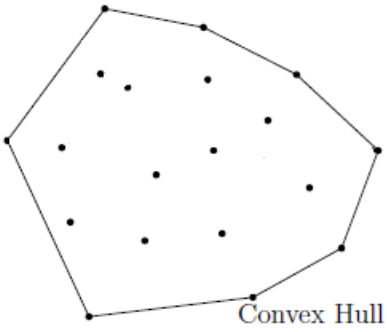

Figure 4 Convex Hull Result

3) Geometry farthest pair

Definition: Given a set of points P, the farthest pair is the pair of points that have the largest Euclidean distance between them.

Example: Figure 5 shows the two points contributing to the farthest pair have to lie on the convex hull. The input of the farthest pair operation is a set of points and the output is a pair of points which have the farthest distances between each other.

Function Name: *GeometryFarthestPair*

Arguments:

(1) *String InputLocation*: the location of the input in HDFS

(2) *String OutputLocation*: the location of the output in HDFS

Requirement:

Load a set of points, output the farthest pair of this set.

Input Dataset Schema:

x, y

// Every row is a point (longitude, latitude). This dataset has a bunch of points.

Output Dataset Schema:

x, y

// Every row is a point (longitude, latitude). This dataset has two points. The farthest pair is composed of them.

4) Geometry closest pair

Definition: Given a set of points P, the closest pair is the pair of points that have the smallest Euclidean distance between them. This pair of points should lie in the convex hull.

Example: Figure 5 shows the two points contributing to the closest pair have to lie on the convex hull. The input of the farthest pair operation is a set of points and the output is a pair of points which have the closest distances between each other.

Function Name: *GeometryClosestPair*

Arguments:

(1) *String InputLocation*: the location of the input in HDFS

(2) *String OutputLocation*: the location of the output in HDFS

Requirement:

Load a set of points, output the closest pair of this set.

Input Dataset Schema:

x, y

// Every row is a point (longitude, latitude). This dataset has a bunch of points.

Output Dataset Schema:

x, y

// Every row is a point (longitude, latitude). This dataset has two points. The closest pair is composed of them.

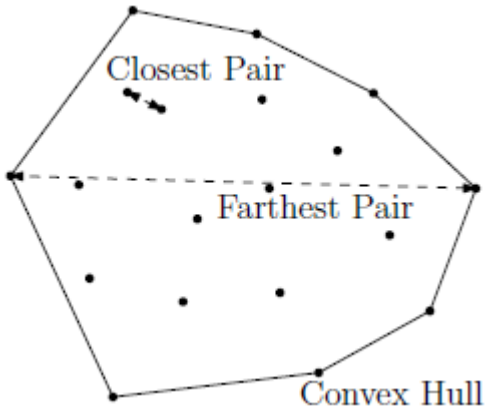

Figure 5 Farthest Pair and Closest Pair in Convex Hull

5) Spatial range query

Definition: Spatial range queries are queries that inquire about certain spatial objects which lie in a certain query window. Here we are interested in finding objects within a rectangular window in space. Only objects falling inside this window are displayed.

Example: The figures below show an example of rectangular window query. The input1 is a set of points, the input2 is a rectangle, and the output is a set of points which are all inside the rectangle.

Function Name: *SpatialRangeQuery*

Arguments:

(1) *String InputLocation1*: the location of the input1 in HDFS

(2) *String InputLocation2*: the location of the input2 in HDFS

(3) *String OutputLocation*: the location of the output in HDFS

Requirement:

Load a set of polygons, output the query result of this set.

Input1 Dataset Schema:

id, x1, y1

// Every row represent one point (longitude, latitude). This set has a bunch of points.

Input2 Dataset Schema:

x1, y1, x2, y2

// This dataset has a pair of points (longitude, latitude) which defines a rectangle. This dataset is the query window of rang query.

Output Dataset Schema:

id

// Every row is the id of a point. This set has a bunch of ids.

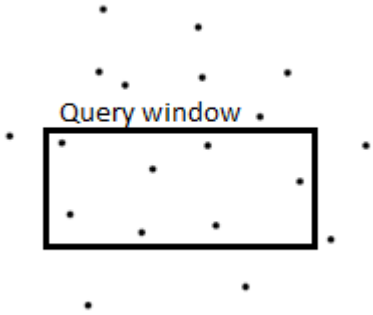

Figure 6 Range Query Input *Figure 7 Range Query Output*

6) Spatial join query

Definition: Spatial join operation is used to combine two or more datasets with respect to a spatial predicate. A typical example of a spatial join query is "Find all pair of rivers and cities that intersect".

Example: The figures below show an example of a join query. The input1 is a set of points, the input2 is a set of polygons. The output is the list of the points and the rectangles which contain them.

Function Name: *SpatialJoinQuery*

Arguments:

(1) *String InputLocation1*: the location of the input 1 in HDFS

(2) *String InputLocation2*: the location of the input 2 in HDFS

(3) *String OutputLocation*: the location of the output in HDFS

(4) *String input1Type*: Indicate input 1 is point, or rectangles. ("point", "rectangle")

Requirement:

Load two sets of polygons, output the join query result of this set.

Input1 Dataset Schema:

(1) A-id, x1, y1, x2, y2

// Every row is a pair of points (longitude, latitude) which defines a rectangle. This set has a bunch of rectangle. Both contain and overlap will be seen as contained, i.e., both should be output to the result.

(2) A-id, x1, y1

//Every row is a pair (longitude, latitude) which defines a point. This set has a bunch of points. You will consider those points on the line of the rectangle in input 2 as "contained", i.e., output to the join result.

Input2 Dataset Schema:

B-id, x1, y1, x2, y2

// Every row is a pair of points (longitude, latitude) which defines a rectangle. This set has a bunch of rectangle.

Output Dataset Schema:

B-id, A-id$_1$, A-id$_2$, A-id$_3$, … A-id$_n$

// Every row has one B-id and 0 ~ n Bid and shows the rectangle (B-id) which contain this points (A-id) or recatangles. This dataset has a bunch of rows.

If the join set is empty: you should output:
Aid, NULL.

Note: The first input data set will be either completely consists of point or completely consists of rectangles.

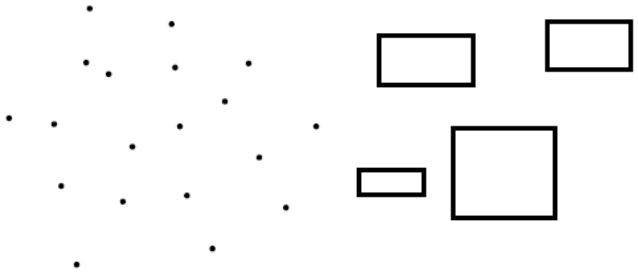

Figure 8 (a) Join Query Input points

Figure 8 (b) Join Query Input Rectangles

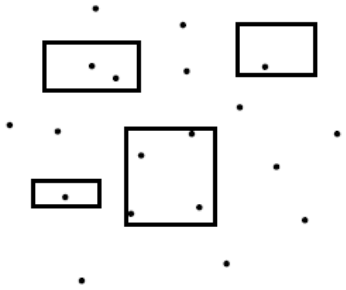

Figure 8 (c) Join Query Effect

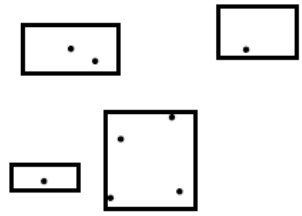

Figure 9 Join Query Result

Some other notes:

(1) Performance matter, if your program can't finish in a reasonable time, you will lose portation of your grade. So optimized your code!

(2) Your result has to be sorted!!!

(3) Keep use RDD, and don't try to collect it, we might want to

(4) Remove all the debug information and keep clean code documentation when submit your code.

(5) By sorted, I mean not only on the key column or the first column in the csv file, I mean for all columns, i.e., output should be like:

1,2 1,3

1,3 but not 1,5

1,5 1,2

(7) Compared to the original requirement, I remove the return value part, you don't need to return anything, just write your output to hdfs.