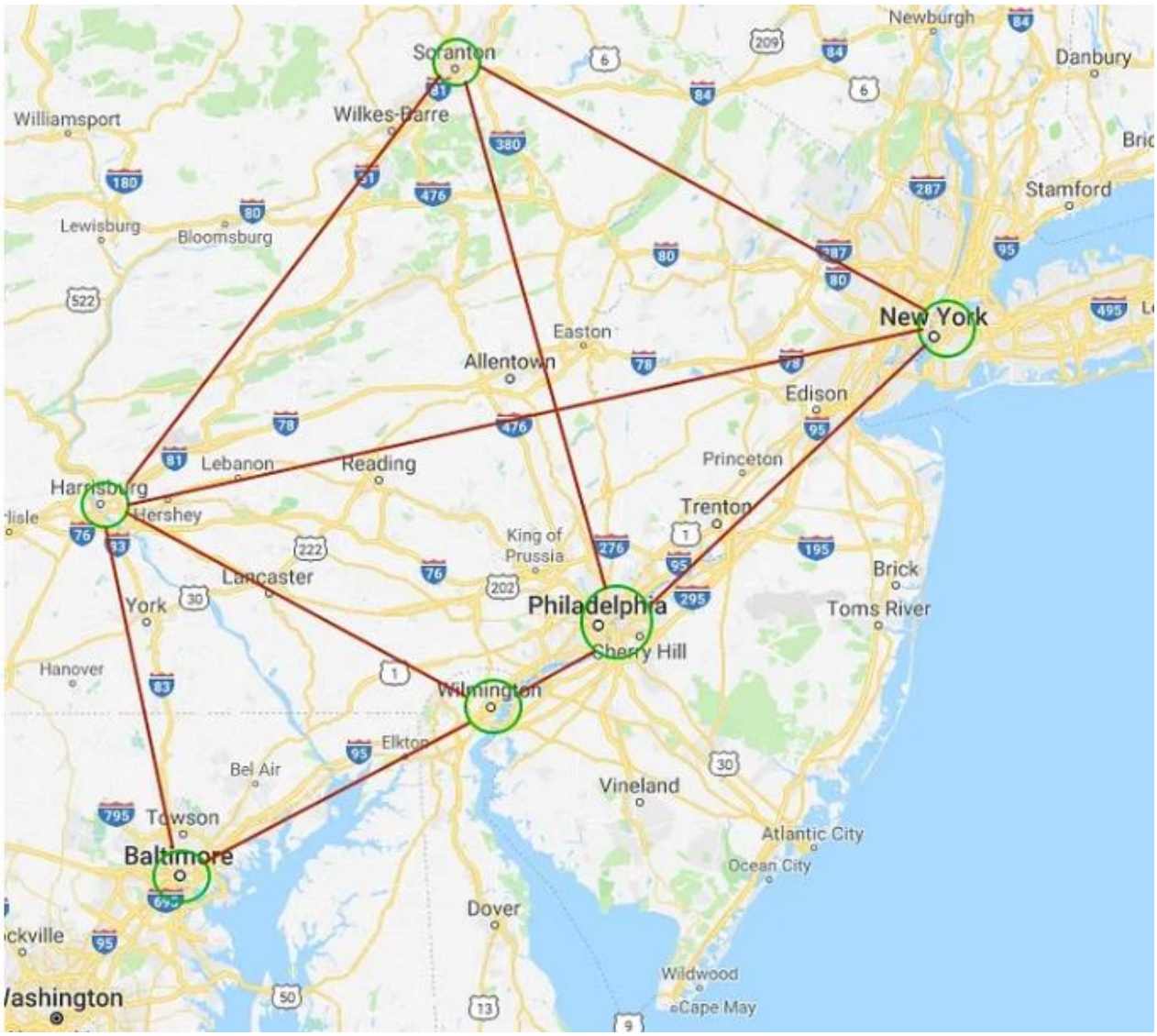
Graphical Data Creation and Access Graphical Data using Spark

We will work on the below graph to understand the creation of graphical data and access of graphical data using spark.



To work with above graph, we need to launch Spark shell.

Step1: Start Cloudera , type in the terminal:
spark-shell

Step2: Now, we have to make some imports:
import org.apache.spark.graphx.Edge #working with edge attribute

import org.apache.spark.graphx.Graph #analyzing and processing large graphs in a distributed fashion

import org.apache.spark.graphx.lib._ #Various analytics functions for graphs.

Step3: To create property graph we should firstly create an array of vertices and an array of edges. For vertices array, type in your spark shell:
val verArray = Array(
(1L, ("Philadelphia", 1580863)),
(2L, ("Baltimore", 620961)),
(3L, ("Harrisburg", 49528)),
(4L, ("Wilmington", 70851)),
(5L, ("New York", 8175133)),
(6L, ("Scranton", 76089)))

The attributes of the vertices mean the city name and population, respectively.

As an output you will see the following:
verArray: Array[(Long, (String, Int))] = Array((1,(Philadelphia,1580863)), (2,(Baltimore,620961)), (3,(Harrisburg,49528)), (4,(Wilmington,70851)), (5,(New York,8175133)), (6,(Scranton,76089)))

Step4: To create edges array, type in the spark shell:
val edgeArray = Array(
Edge(2L, 3L, 113),
Edge(2L, 4L, 106),
Edge(3L, 4L, 128),
Edge(3L, 5L, 248),
Edge(3L, 6L, 162),
Edge(4L, 1L, 39),
Edge(1L, 6L, 168),
Edge(1L, 5L, 130),
Edge(5L, 6L, 159))

The first and the second arguments indicate the source and the destination vertices identifiers and the third argument means the edge property which, in our case, is the distance between corresponding cities in kilometres.

The above-mentioned input will give us the following output:
edgeArray: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(2,3,113), Edge(2,4,106), Edge(3,4,128), Edge(3,5,248), Edge(3,6,162), Edge(4,1,39), Edge(1,6,168), Edge(1,5,130), Edge(5,6,159))

Step5: Next, we will create RDDs from the vertices and edges arrays by using the sc.parallelize()command:

Note: Resilient Distributed Dataset (RDD) is the fundamental data structure of Spark. They are immutable Distributed collections of objects of any type. As the name suggests is a Resilient (Fault-tolerant) records of data that resides on multiple nodes

val verRDD = sc.parallelize(verArray)

You will see:
verRDD: org.apache.spark.rdd.RDD[(Long, (String, Int))] = ParallelCollectionRDD[0] at parallelize at <console>:34

val edgeRDD = sc.parallelize(edgeArray)

You will see:
edgeRDD:org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelCollectionRDD[1] at parallelize at <console>:34

Step6: We are ready to build a property graph. The basic property graph constructor takes an RDD of vertices and an RDD of edges and builds a graph.

val graph = Graph(verRDD, edgeRDD)

You will see:
graph: org.apache.spark.graphx.Graph[(String, Int),Int] = org.apache.spark.graphx.impl.GraphImpl@79ce6829

Step7: Now, we have our property graph, and it is time to consider basic operations which can be performed with graphs such as filtration by vertices, filtration by edges and operations with triplets.

A)Filtration by vertices
To illustrate the filtration by vertices let's find the cities with population more than 50000.

To implement this, we will use the filter operator:

```
graph.vertices.filter {
case (id, (city, population)) => population > 50000
}.collect.foreach {
case (id, (city, population)) =>
println(s"The population of $city is $population")
}
```

And this is the result we get:
The population of Scranton is 76089
The population of Wilmington is 70851
The population of Philadelphia is 1580863
The population of New York is 8175133
The population of Baltimore is 620961

B)Triplets

One of the core functionalities of GraphX is exposed through the triplets RDD. There is one triplet for each edge which contains information about both the vertices and the edge information. Let's take a look through graph.triplets.collect.

As an example of working with triplets, we will find the distances between the connected cities:

```
for (triplet <- graph.triplets.collect) {
println(s"""The distance between ${triplet.srcAttr._1} and
${triplet.dstAttr._1} is ${triplet.attr} kilometers""")
}
```

As a result, you should see:

```
The distance between Baltimore and Harrisburg is 113 kilometers
The distance between Baltimore and Wilmington is 106 kilometers
The distance between Harrisburg and Wilmington is 128 kilometers
The distance between Harrisburg and New York is 248 kilometers
The distance between Harrisburg and Scranton is 162 kilometers
The distance between Wilmington and Philadelphia is 39 kilometers
The distance between Philadelphia and New York is 130 kilometers
The distance between Philadelphia and Scranton is 168 kilometers
The distance between New York and Scranton is 159 kilometers
```

C)Filtration by edges

Now, let's consider another type of filtration, namely filtration by edges. For this purpose, we want to find the cities, the distance between which is less than 150 kilometers. If we type in the spark shell,

```
graph.edges.filter {
case Edge(city1, city2, distance) => distance < 150
}.collect.foreach {
case Edge(city1, city2, distance) =>
println(s"The distance between $city1 and $city2 is $distance")
}
```

The result will be:

```
The distance between 2 and 3 is 113
The distance between 2 and 4 is 106
The distance between 3 and 4 is 128
The distance between 4 and 1 is 39
The distance between 1 and 5 is 130
```