Hands On: Building a Graph Reading | Coursera

Hands On: Building A Graph

1. Start the Cloudera VM and Upload the datasets to HDFS
2. Import the GraphX libraries
3. Import the Vertices
4. Import the Edges
5. Create a Graph
6. Use Spark's filter method to return Vertices in the graph

Start the Cloudera VM and Upload the datasets to HDFS

Ensure the Cloudera Quick Start virtual machine is started and that you have downloaded ExamplesOfAnalytics.zip from the provided link in the content section for this week.

Copy the ExamplesOfAnalytics.zip file to the Cloudera's Home folder before proceeding.

Open a terminal in the Cloudera Quick Start virtual machine by clicking **Applications**, **System Tools** then **Terminal**.

Type the following command in the Terminal window to extract the zip file to the Cloudera Home directory.

```
unzip ExamplesOfAnalytics.zip
```

Type the following command in the Terminal window to go into the ExamplesOfAnalytics directory.

```
cd ExamplesOfAnalytics
```

Use the hdfs command to upload the datasets in the EOADATA directory to HDFS.

```
hdfs dfs -put EOADATA
```

Start the Spark Shell will all of the libraries needed to complete the hands on exercises.

```
spark-shell --jars lib/gs-core-1.2.jar,lib/gs-ui-1.2.jar,lib/jcommon-
1.0.16.jar,lib/jfreechart-1.0.13.jar,lib/breeze_2.10-0.9.jar,lib/breeze-
viz_2.10-0.9.jar,lib/pherd-1.0.jar
```

It may take several seconds for the Spark Shell to start. Be patient and wait for the **scala>** prompt.

Import the GraphX libraries

Set log level to error, suppress info and warn messages.

```
import org.apache.log4j.Logger
import org.apache.log4j.Level

Logger.getLogger("org").setLevel(Level.ERROR)
Logger.getLogger("akka").setLevel(Level.ERROR)
```

Import the Spark's GraphX and RDD libraries along with Scala's source library.

```
import org.apache.spark.graphx._
import org.apache.spark.rdd._

import scala.io.Source
```

Import the Vertices

Before importing any datasets, let view what the files contain. Print the first 5 lines of each comma delimited text file.

input:

```
Source.fromFile("./EOADATA/metro.csv").getLines().take(5).foreach(println)
```

output:

```
#metro_id,name,population
 1,Tokyo,36923000
 2,Seoul,25620000
 3,Shanghai,24750000
 4,Guangzhou,23900000
```

input:

```
Source.fromFile("./EOADATA/country.csv").getLines().take(5).foreach(println)
```

output:

```
#country_id,name
1,Japan
2,South Korea
3,China
4,India
```

input:

```
Source.fromFile("./EOADATA/metro_country.csv").getLines().take(5).foreach(pri
ntln)
```

output:

```
#metro_id,country_id
1,1
2,2
3,3
4,3
```

Create case classes for the places (metros and countries).

input:

```
class PlaceNode(val name: String) extends Serializable
```

output:

```
defined class PlaceNode
```

input:

```
case class Metro(override val name: String, population: Int) extends
PlaceNode(name)
```

output:

```
defined class Metro
```

input:

```
case class Country(override val name: String) extends PlaceNode(name)
```

output:

```
defined class Country
```

Read the comma delimited text file metros.csv into an RDD of Metro vertices, ignore lines that start with # and map the columns to: id, Metro(name, population).

input:

```
val metros: RDD[(VertexId, PlaceNode)] =
  sc.textFile("./EOADATA/metro.csv").
    filter(! _.startsWith("#")).
    map {line =>
      val row = line split ','
      (0L + row(0).toInt, Metro(row(1), row(2).toInt))
    }
```

output:

```
metros: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId,
PlaceNode)] = MapPartitionsRDD[18] at map at <console>:36
```

Read the comma delimited text file country.csv into an RDD of Country vertices, ignore lines that start with # and map the columns to: id, Country(name). Add 100 to the country indexes so they are unique from the metro indexes.

input:

```
val countries: RDD[(VertexId, PlaceNode)] =
  sc.textFile("./EOADATA/country.csv").
    filter(! _.startsWith("#")).
    map {line =>
      val row = line split ','
      (100L + row(0).toInt, Country(row(1)))
    }
```

output:

```
countries: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId,
PlaceNode)] = MapPartitionsRDD[26] at map at <console>:36
```

Import the Vertices

Read the comma delimited text file metro_country.tsv into an RDD[Edge[Int]] collection. Remember to add 100 to the countries' vertex id.

input:

```
val mclinks: RDD[Edge[Int]] =
  sc.textFile("./EOADATA/metro_country.csv").
    filter(! _.startsWith("#")).
    map {line =>
      val row = line split ','
      Edge(0L + row(0).toInt, 100L + row(1).toInt, 1)
    }
```

output:

```
mclinks: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] =
MapPartitionsRDD[30] at map at <console>:33
```

Create a Graph

Concatenate the two sets of nodes into a single RDD.

input:

```
val nodes = metros ++ countries
```

output:

```
nodes: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId,
PlaceNode)] = UnionRDD[31] at $plus$plus at <console>:39
```

Pass the concatenated RDD to the Graph() factory method along with the RDD link input:

```
val metrosGraph = Graph(nodes, mclinks)
```

output:

```
metrosGraph: org.apache.spark.graphx.Graph[PlaceNode,Int] =
org.apache.spark.graphx.impl.GraphImpl@7b13f4ea
```

Print the first 5 vertices and edges.

input:

```
metrosGraph.vertices.take(5)
```

output:

```
res8: Array[(org.apache.spark.graphx.VertexId, PlaceNode)] =
Array((34,Metro(Hong Kong,7298600)),
(52,Metro(Ankara,5150072)), (4,Metro(Guangzhou,23900000)),
(16,Metro(Istanbul,14377018)), (28,Metro(Nagoya,9107000)))
```

input:

```
metrosGraph.edges.take(5)
```

output:

```
res9: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,101,1),
Edge(2,102,1), Edge(3,103,1), Edge(4,103,1), Edge(5,104,1))
```

Use Spark’s filter method to return Vertices in the graph

Filter all of the edges in metrosGraph that have a source vertex Id of 1 and create a map of destination vertex Ids.

input:

```
metrosGraph.edges.filter(_.srcId == 1).map(_.dstId).collect()
```

output:

```
res10: Array[org.apache.spark.graphx.VertexId] = Array(101)
```

Similarly, filter all of the edges in metrosGraph where the destination vertexId is 103 and create a map of all of the source Ids.

input:

```
metrosGraph.edges.filter(_.dstId == 103).map(_.srcId).collect()
```

output:

```
res11: Array[org.apache.spark.graphx.VertexId] = Array(3, 4, 7, 24, 34)
```