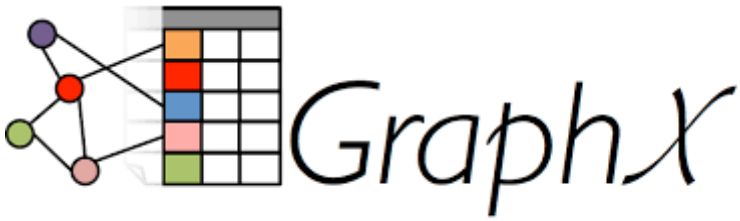
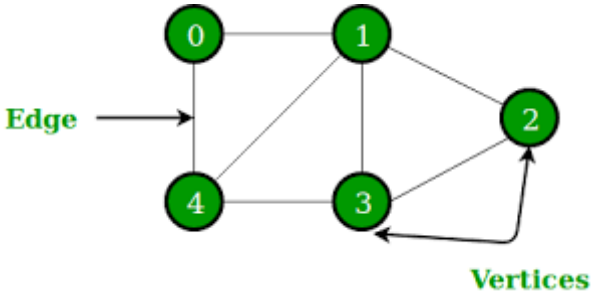
Apache GraphX

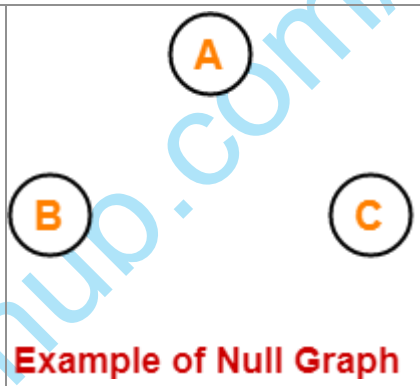


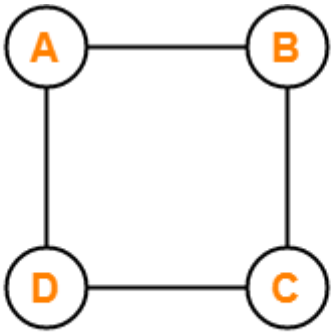
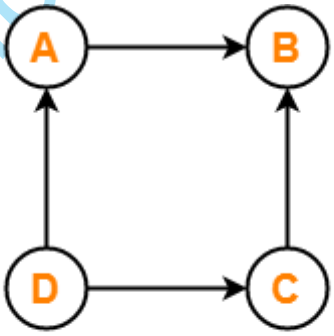
A graph is made up of two sets called Vertices and Edges.

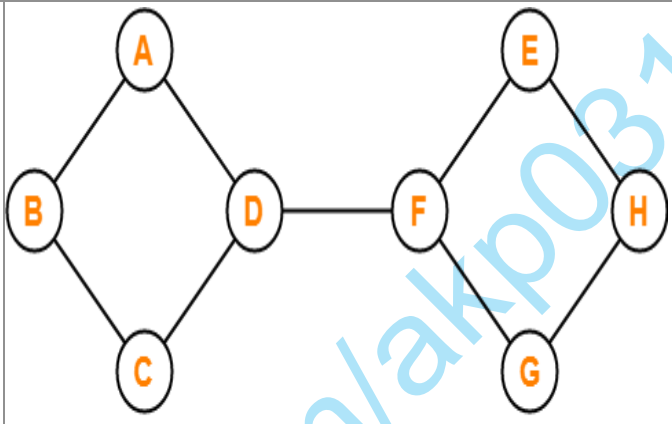


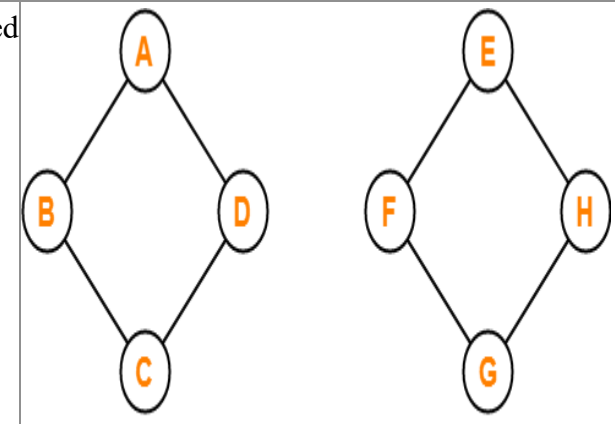
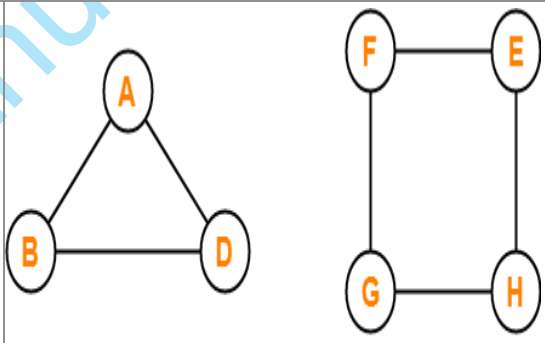
- Vertex is the fundamental unit of which graphs are formed.
- Pair of vertices that specifies a line joining a two vertices represents an Edge.

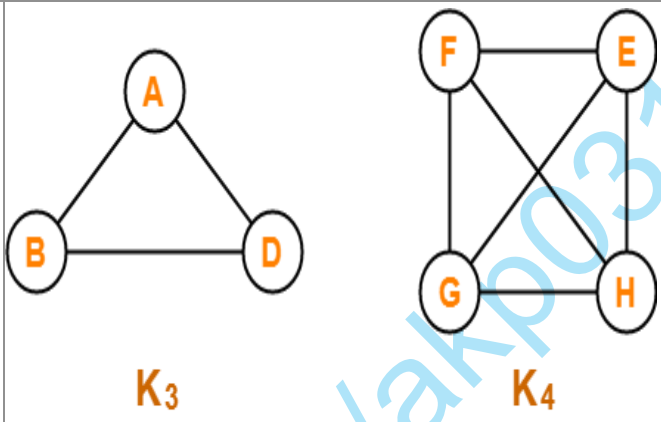
Types of Graphs

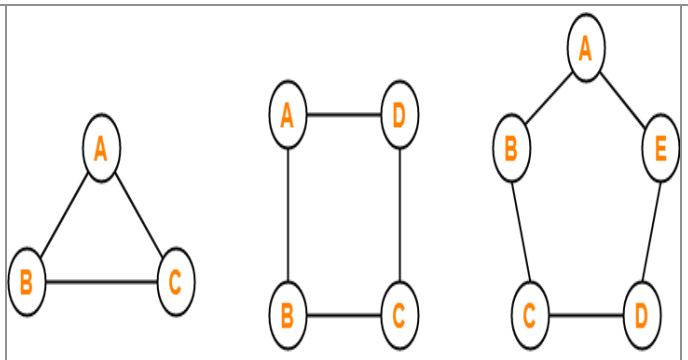
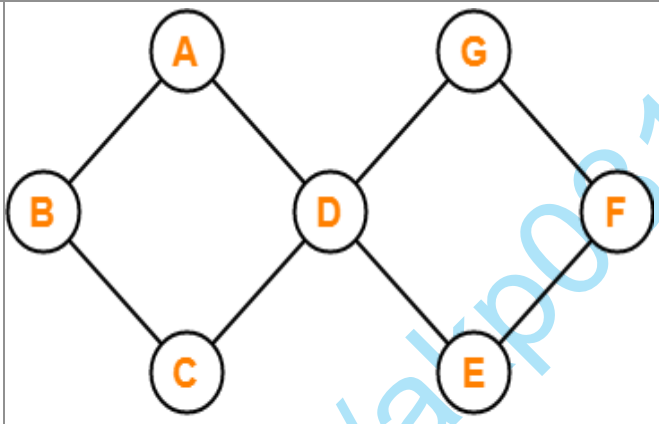
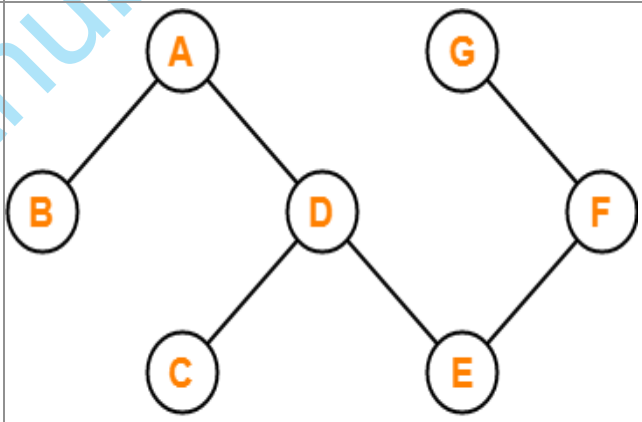
Null Graph		• A graph whose edge set is empty is called as a null graph. • In other words, a null graph does not contain any edges in it. • This graph consists only of the vertices and there are no edges in it. • Since the edge set is empty, therefore it is a null graph.

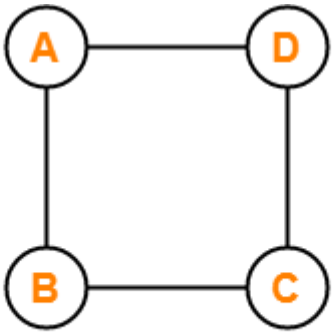
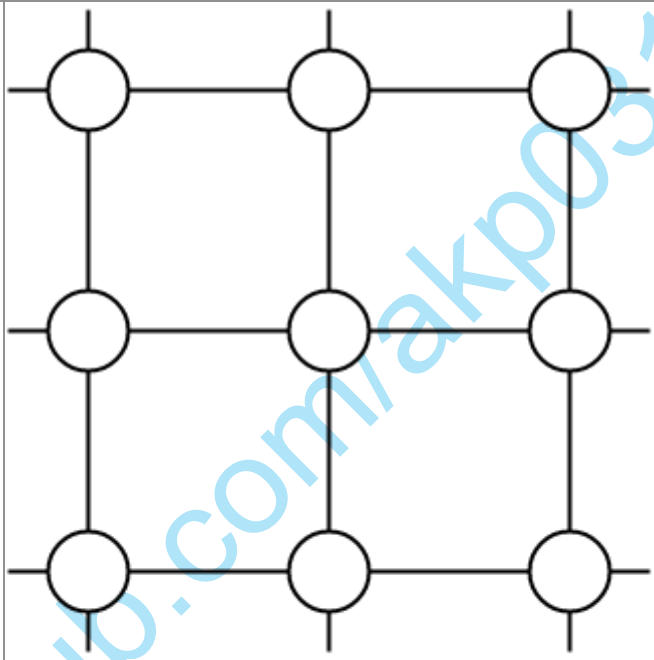
Undirected Graph or Non-Directed Graph	 Example of Non-Directed Graph	• A graph in which all the edges are undirected is called as a non-directed graph. • In other words, edges of an undirected graph do not contain any direction. Example: • This graph consists of four vertices and four undirected edges. • Since all the edges are undirected, therefore it is a non-directed graph.
Directed Graph	 Example of Directed Graph	• A graph in which all the edges are directed is called as a directed graph. • In other words, all the edges of a directed graph contain some direction. • Directed graphs are also called as digraphs.

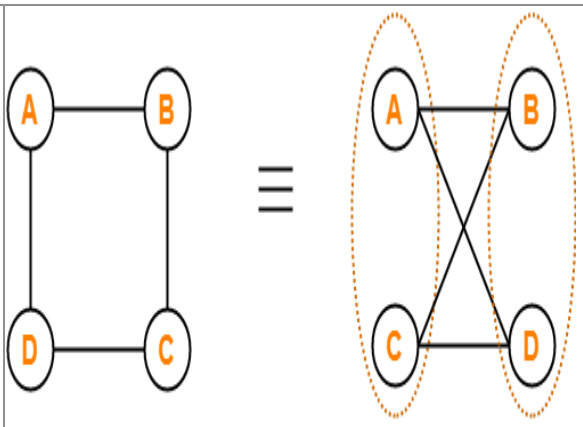
		Example: • This graph consists of four vertices and four directed edges. • Since all the edges are directed, therefore it is a directed graph.
Connected Graph	 Example of Connected Graph	• A graph in which we can visit from any one vertex to any other vertex is called as a connected graph. • In connected graph, at least one path exists between every pair of vertices. Example • In this graph, we can visit from any one vertex to any other vertex. • There exists at least one path between every pair of vertices. • Therefore, it is a connected graph.

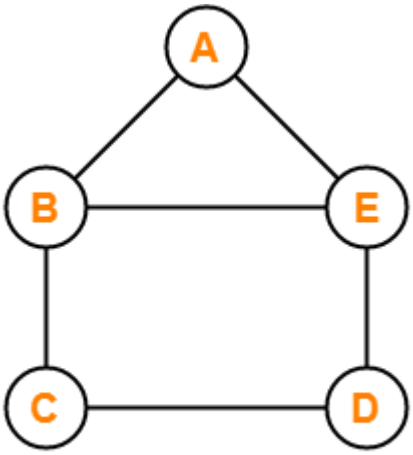
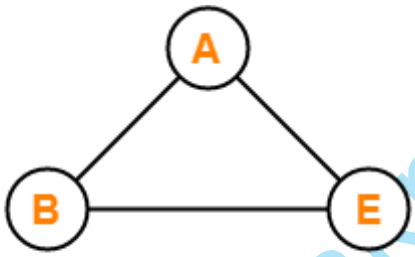
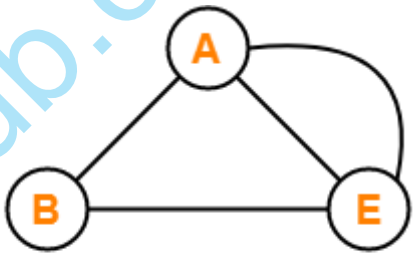
Disconnected Graph	 Example of Disconnected Graph	• A graph in which there does not exist any path between at least one pair of vertices is called as a disconnected graph. Example • This graph consists of two independent components which are disconnected. • It is not possible to visit from the vertices of one component to the vertices of other component. • Therefore, it is a disconnected graph.
Regular Graph	 Examples of Regular Graph	• A graph in which degree of all the vertices is same is called as a regular graph. • If all the vertices in a graph are of degree ‘k’, then it is called as a “k-regular graph“.

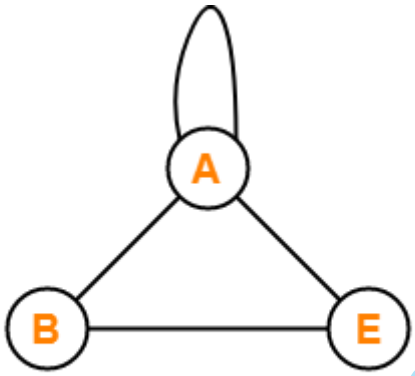
		Example In these graphs, • All the vertices have degree-2. • Therefore, they are 2-Regular graphs.
Complete Graph	 Examples of Complete Graph	• A graph in which exactly one edge is present between every pair of vertices is called as a complete graph. • A complete graph of ‘n’ vertices contains exactly nC2 edges. • A complete graph of ‘n’ vertices is represented as Kn. Example • Each vertex is connected with all the remaining vertices through exactly one edge. • Therefore, they are complete graphs.

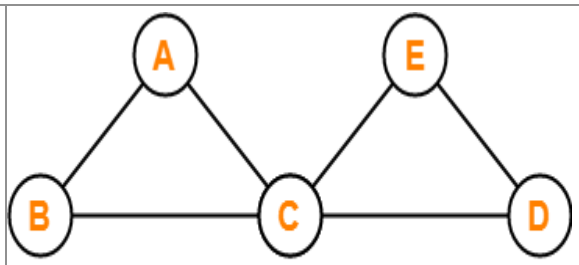
Cycle Graph	 Examples of Cycle Graph	• A simple graph of 'n' vertices (n>=3) and n edges forming a cycle of length 'n' is called as a cycle graph. • In a cycle graph, all the vertices are of degree 2.
Cyclic Graph	 Example of Cyclic Graph	• A graph containing at least one cycle in it is called as a cyclic graph. Example • This graph contains two cycles in it. • Therefore, it is a cyclic graph.
Acyclic Graph	 Example of Acyclic Graph	• A graph not containing any cycle in it is called as an acyclic graph.

Finite Graph	 Example of Finite Graph	• A graph consisting of finite number of vertices and edges is called as a finite graph.
Infinite Graph	 Example of Infinite Graph	• A graph consisting of infinite number of vertices and edges is called as an infinite graph.

Bipartite Graph	 Example of Bipartite Graph	A bipartite graph is a special kind of graph with the following properties- • It consists of two sets of vertices X and Y. • The vertices of set X join only with the vertices of set Y. • The vertices within the same set do not join. Here, • The vertices of the graph can be decomposed into two sets. • The two sets are X = {A, C} and Y = {B, D}. • The vertices of set X join only with the vertices of set Y and vice-versa. • The vertices within the same set do not join. • Therefore, it is a bipartite graph.

Planar Graph	 Example of Planar Graph	• A planar graph is a graph that we can draw in a plane such that no two edges of it cross each other.
Simple Graph	 Example of Simple Graph	• A graph having no self-loops and no parallel edges in it is called as a simple graph.
Multi Graph	 Example of Multi Graph	• A graph having no self-loops but having parallel edge(s) in it is called as a multi graph. Here, • This graph consists of three vertices and four edges out

		of which one edge is a parallel edge. • There are no self-loops but a parallel edge is present. • Therefore, it is a multi-graph.
Pseudo Graph	 Example of Pseudo Graph	• A graph having no parallel edges but having self-loop(s) in it is called as a pseudo graph. Here, • This graph consists of three vertices and four edges out of which one edge is a self-loop. • There are no parallel edges but a self-loop is present. • Therefore, it is a pseudo graph.

Euler Graph	 Example of Euler Graph	Euler Graph is a connected graph in which all the vertices are even degree. Here, • This graph is a connected graph. • The degree of all the vertices is even. • Therefore, it is an Euler graph.

Graph: Imagine a graph as a collection of points (nodes) connected by lines (edges). Each point represents something (like a person, place, or thing), and each line represents a relationship between those things.

Property Graph: Now, think of a property graph as a graph where both the points (nodes) and the lines (edges) can have additional information attached to them. This extra information is called properties.

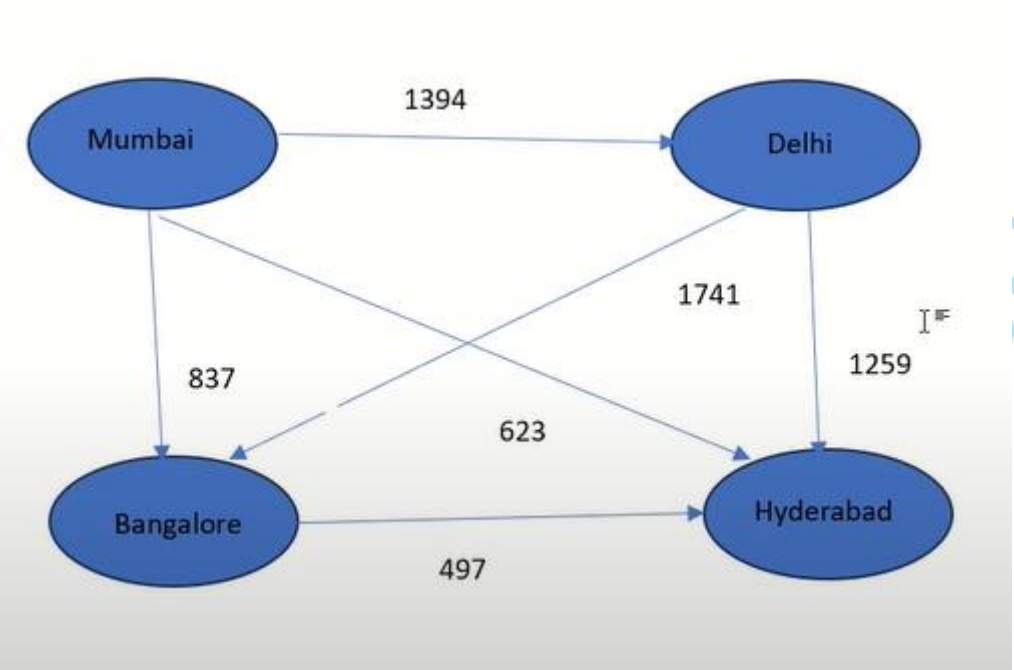
Apache GraphX: This is a graph processing framework built on top of Apache Spark, a powerful tool for big data processing. GraphX allows us to work with massive graphs efficiently, performing operations like traversing the graph, analyzing its structure, and running algorithms on it.

To get started first we need to import Spark and GraphX into our project, as follows:

```
import org.apache.spark._
```

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

import org.apache.spark.rdd.RDD



Steps	Command
Step-1	import org.apache.spark.graphx.Edge import org.apache.spark.graphx.Graph import org.apache.spark.graphx.lib._

```
scala> import org.apache.spark.graphx.Edge
import org.apache.spark.graphx.Edge

scala> import org.apache.spark.graphx.Graph
import org.apache.spark.graphx.Graph

scala> import org.apache.spark.graphx.lib._
import org.apache.spark.graphx.lib._
```

Step-2

```
val verArray = Array(
(1L, ("Mumbai", 12442373)),
(2L, ("Delhi", 11034555)),
(3L, ("Bangalore", 8443675)),
(4L, ("Hyderabad", 6993262))
)
```

```
scala> val verArray = Array(
     | (1L, ("Mumbai", 12442373)),
     | (2L, ("Delhi", 11034555)),
     | (3L, ("Bangalore", 8443675)),
     | (4L, ("Hyderabad", 6993262))
     | )
verArray: Array[(Long, (String, Int))] = Array((1,(Mumbai,12442373)), (2,(Delhi,11034555)), (3,(Bangalore,8443675)), (4,(Hyderabad,6993262)))
```

Step-3

```
val edgeArray = Array(
Edge(1L, 2L, 1394),
Edge(1L, 3L, 837),
Edge(1L, 4L, 623),
Edge(2L, 3L, 1741),
Edge(2L, 4L, 1259),
Edge(3L, 4L, 497)
)
```

```
scala> val edgeArray = Array(
     | Edge(1L, 2L, 1394),
     | Edge(1L, 3L, 837),
     | Edge(1L, 4L, 623),
     | Edge(2L, 2L, 1741),
     | Edge(2L, 4L, 1259),
     | Edge(3L, 4L, 497)
     | )
edgeArray: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1394), Edge(1,3,837), Edge(1,4,623), Edge(2,2,1741), Edge(2,4,1259), Edge(3,4,497))
```

Step-4

val verRDD = sc.parallelize(verArray)

val edgeRDD = sc.parallelize(edgeArray)

```
scala> val verRDD = sc.parallelize(verArray)
verRDD: org.apache.spark.rdd.RDD[(Long, (String, Int))] = ParallelCollectionRDD[0] at parallelize at <console>:29

scala> val edgeRDD = sc.parallelize(edgeArray)
edgeRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelCollectionRDD[1] at parallelize at <console>:29
```

Step-5

val graph = Graph(verRDD, edgeRDD)

```
scala> val graph = Graph(verRDD, edgeRDD)
graph: org.apache.spark.graphx.Graph[(String, Int),Int] = org.apache.spark.graphx.impl.GraphImpl@5f0469e2
```

Step-6

graph.numVertices

graph.numEdges

graph.inDegrees.collect()

graph.outDegrees.collect()

graph.degrees.collect()

```
scala> graph.numVertices
res0: Long = 4

scala> graph.numEdges
res1: Long = 6

scala> graph.inDegrees.collect()
res2: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,3), (2,2), (3,1))

scala> graph.outDegrees.collect()
res3: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((1,3), (2,2), (3,1))

scala> graph.degrees.collect()
res4: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,3), (1,3), (2,4), (3,
```

Step-7

graph.vertices.collect.foreach(println)

graph.edges.collect.foreach(println)

graph.triplets.collect.foreach(println)

```
scala> graph.vertices.collect.foreach(println)
(4,(Hyderabad,6993262))
(1,(Mumbai,12442373))
(2,(Delhi,11034555))
(3,(Bangalore,8443675))

scala> graph.edges.collect.foreach(println)
Edge(1,2,1394)
Edge(1,3,837)
Edge(1,4,623)
Edge(2,2,1741)
Edge(2,4,1259)
Edge(3,4,497)

scala> graph.triplets.collect.foreach(println)
((1,(Mumbai,12442373)),(2,(Delhi,11034555)),1394)
((1,(Mumbai,12442373)),(3,(Bangalore,8443675)),837)
((1,(Mumbai,12442373)),(4,(Hyderabad,6993262)),623)
((2,(Delhi,11034555)),(2,(Delhi,11034555)),1741)
((2,(Delhi,11034555)),(4,(Hyderabad,6993262)),1259)
((3,(Bangalore,8443675)),(4,(Hyderabad,6993262)),497)
```

Step-8	val triplets = graph.triplets
	
	triplets.count()
	
	`scala> val triplets = graph.triplets`
	`triplets: org.apache.spark.rdd.RDD[org.apache.spark.graphx.EdgeTriplet[(String, Int),Int]] = MapPartitionsRDD[33] at mapPartitions at GraphImpl.scala:47`
	
	`scala> triplets.count()`
	`res8: Long = 6`

Step-9	graph.vertices.filter{case (id, (city, population)) => population > 8000000}.collect()
	
	graph.edges.filter{case Edge(city1, city2, distance) => distance < 700}.collect()
	
	`scala> graph.vertices.filter{case (id, (city, population)) => population > 8000000}.collect()`
	`res9: Array[(org.apache.spark.graphx.VertexId, (String, Int))] = Array((1,(Mumbai,12442373)), (2,(Delhi,11034555)), (3,(Bangalore,8443675)))`
	
	`scala> graph.edges.filter{case Edge(city1, city2, distance) => distance < 700}.collect()`
	`res10: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,4,623), Edge(3,4,497))`