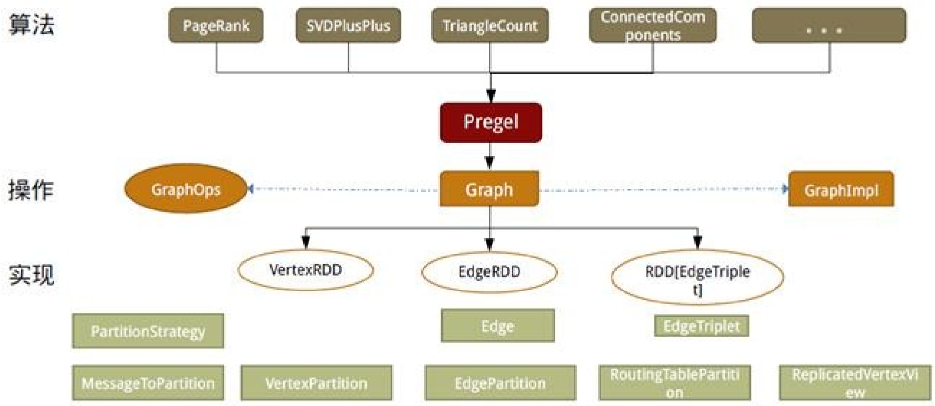
第六节课

Spark GraphX

简介

Pregel是google的大规模图计算系统，spark借鉴了Pregel的架构开发了Bagel，Bagel被废弃开发出了GraphX。

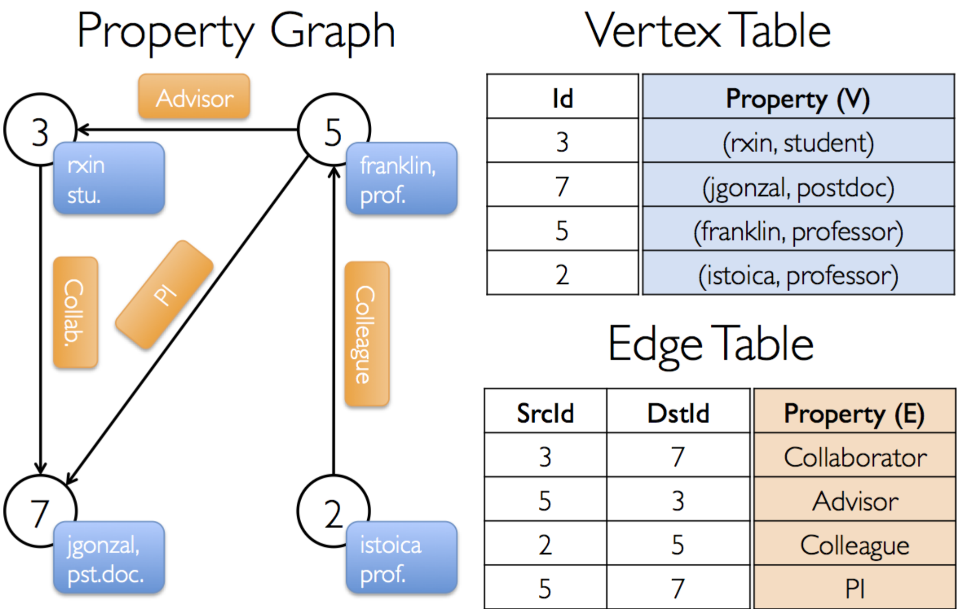


- Spark GraphX是对于带有顶点与边属性的有向多重图进行大规模分布式计算的框架，底层做了大量的优化并提供了丰富的计算分析以及算法操作。
- 属性图是一个有向的多重图，用户为每一个顶点（vertex）和边（edge）定义对象。
- 其中的多重边是相同的源和目的顶点（vertex）之间有不同的属性。支持多重边的能力简化了相同顶点间有多重关系（例如，同事和朋友）的建模场景。
- 每一个顶点以唯一的64位长度标识（vertexId）作为键并且没有对顶点标识符强加一个排序限制。
- 边有对应的源和目的顶点标识符。
- 属性图跟RDD一样是immutable、分布式的和容错的，改变图的结构或值会激发相应的优化措施在内存中产生新的图。

属性图的构成

```
class Graph[VD, ED] {
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
}
```

VertexRDD[VD] 、 EdgeRDD[ED] 分别扩展并优化了RDD[(VertexId, VD)] 、 RDD[Edge[ED]].



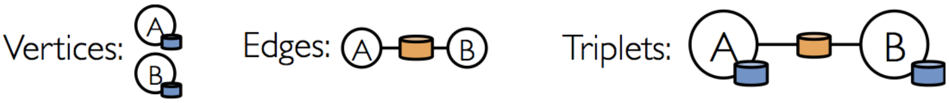
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

属性图的构成



作业

1. 实现求共同好友的算法
2. 为了将图向量化，实现Node2Vec算法

代码

```
/**
  * Created by ding on 2018/1/7.
  */
import org.apache.spark._
import org.apache.spark.graphx._
import org.apache.spark.storage.StorageLevel
// To make some of the examples work we will also need RDD
import org.apache.spark.rdd.RDD
import org.apache.spark.SparkContext
object graphx {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("SparkGraphX").setMaster("local[2]")
    val sc = new SparkContext(conf)
    sc.setLogLevel("WARN")

    println("--------------------了解属性图的结构-------------------------")
    // 定义点结构，以Long类型为键
    val users: RDD[(VertexId, (String, String))] =
      sc.parallelize(Array((3.toLong, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
        (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))
    // 定义边的RDD，可以出现尚未定义的点,可以定义多重边
    val relationships: RDD[Edge[String]] =
      sc.parallelize(Array(Edge(3L, 7L, "collab"), Edge(3L, 7L, "couple"),  Edge(5L, 3L, "advisor"),
        Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))
    //利用点与边生成图
    val graph: Graph[(String, String), String] = Graph(users, relationships)
    //获取图的点结构
    val usersBack: VertexRDD[(String, String)] = graph.vertices
    usersBack.filter { p => p._2 == null }.foreach(println)
    //usersBack.mapValues()
    //获取图的边结构
    val relationshipsBack: EdgeRDD[String] = graph.edges
    relationshipsBack.filter{case Edge(src, dst, prop) => src < dst}.foreach(println)
    //获取图的tripleta结构
    val triplet = graph.triplets
    triplet.filter(_.srcAttr != null).foreach{triplet =>
      println(triplet.srcAttr._1 + " is the " + triplet.attr + " of " + triplet.dstAttr._1)
    }


    println("--------------------了解基本图属性操作-------------------------")
    println("图的边数量"+graph.numEdges)
    println("图的点数量"+graph.numVertices)
    //出度或入度为0的点不会统计
    println("图的出度")
    graph.outDegrees.foreach(println)
    println("图的入度")
    graph.inDegrees.foreach(println)
    println("图的度")
    graph.degrees.foreach(println)
    println("修改边属性")
    graph.mapEdges(edge => if(edge.srcId == 1L) "newRelationShip" else edge.attr).edges.foreach(println)
    println("修改点属性")
    graph.mapVertices{case(id, attr) => if(attr == null) ("spark","God") else attr}.vertices.foreach(println)
    println("修改triplet属性")
    graph.mapTriplets(t => if(t.srcAttr == null) "newRelationShip" else t.attr).triplets.foreach(println)



    println("--------------------了解基本图结构操作-------------------------")
    println("图反向")
    graph.reverse.edges.foreach(println)
    println("求子图")
    val subGraph= graph.subgraph(vpred = (id, attr) => attr._2 == "prof")
    subGraph.vertices.foreach(println)
    println("mask")
    graph.connectedComponents().mask(subGraph).edges.foreach(println)
    println("相同点间的多重边聚合")
    graph.groupEdges((a,b) => a.concat(b)).edges.foreach(println)
    println("Join操作")
    val outDegrees: VertexRDD[Int] = graph.outDegrees
    graph.outerJoinVertices(outDegrees) { (id, oldAttr, outDegOpt) =>
      outDegOpt match {
        case Some(outDeg) => outDeg
        case None => 0 // No outDegree means zero outDegree
      }
    }.vertices.foreach(println)



    println("--------------------了解基本图邻居操作-------------------------")
    println("消息聚合")
    //基于标签传播的社区发现 一度二度人脉 共同好友
    /*def aggregateMessages[Msg: ClassTag](
                                       sendMsg: EdgeContext[VD, ED, Msg] => Unit,
                                       mergeMsg: (Msg, Msg) => Msg,
                                       tripletFields: TripletFields = TripletFields.All
                                       )
      : VertexRDD[Msg]
    */
    val aggregateGraph  = graph.aggregateMessages[(Long)](
      triplet => { // Map Function
        if (triplet.srcId > triplet.dstId) {
          // Send message to destination vertex containing counter and age
          triplet.sendToDst(triplet.srcId)
        }
      },
      // Add counter and age
```

```
      (a, b) => (a + b)
    )
    aggregateGraph.foreach(println)

    println("求图点的邻居结点")
     graph.collectNeighborIds(EdgeDirection.Out).collect().foreach{case(id, array) =>
       println(id + "邻居： ")
       array.foreach(println)
     }
    println("求图点的邻居")
    graph.collectNeighbors(EdgeDirection.Out).collect().foreach{case(id, array) =>
      println(id + "邻居： ")
      array.foreach(l=>println(l._2))
    }

    println("--------------------了解基本图算法-------------------------")
    graph.edges.foreach(println)
    println("pageRank")
    graph.pageRank(0.0001).vertices.foreach(println)
    println("连通图")
    //社区划分
    graph.connectedComponents().vertices.foreach(println)
    println("三角计数")
    //分析一个人的社区关系
    graph.triangleCount().vertices.foreach(println)
  }
}
```