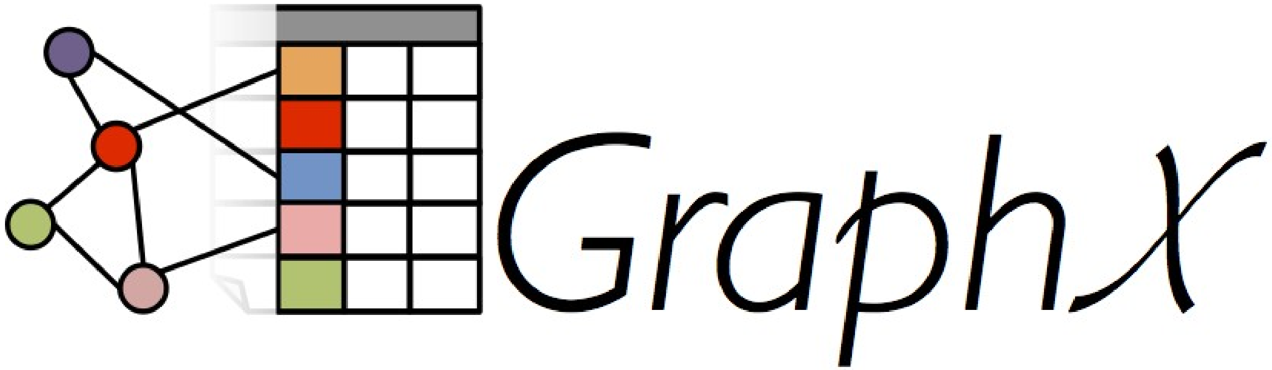


第七课： GraphX 入门

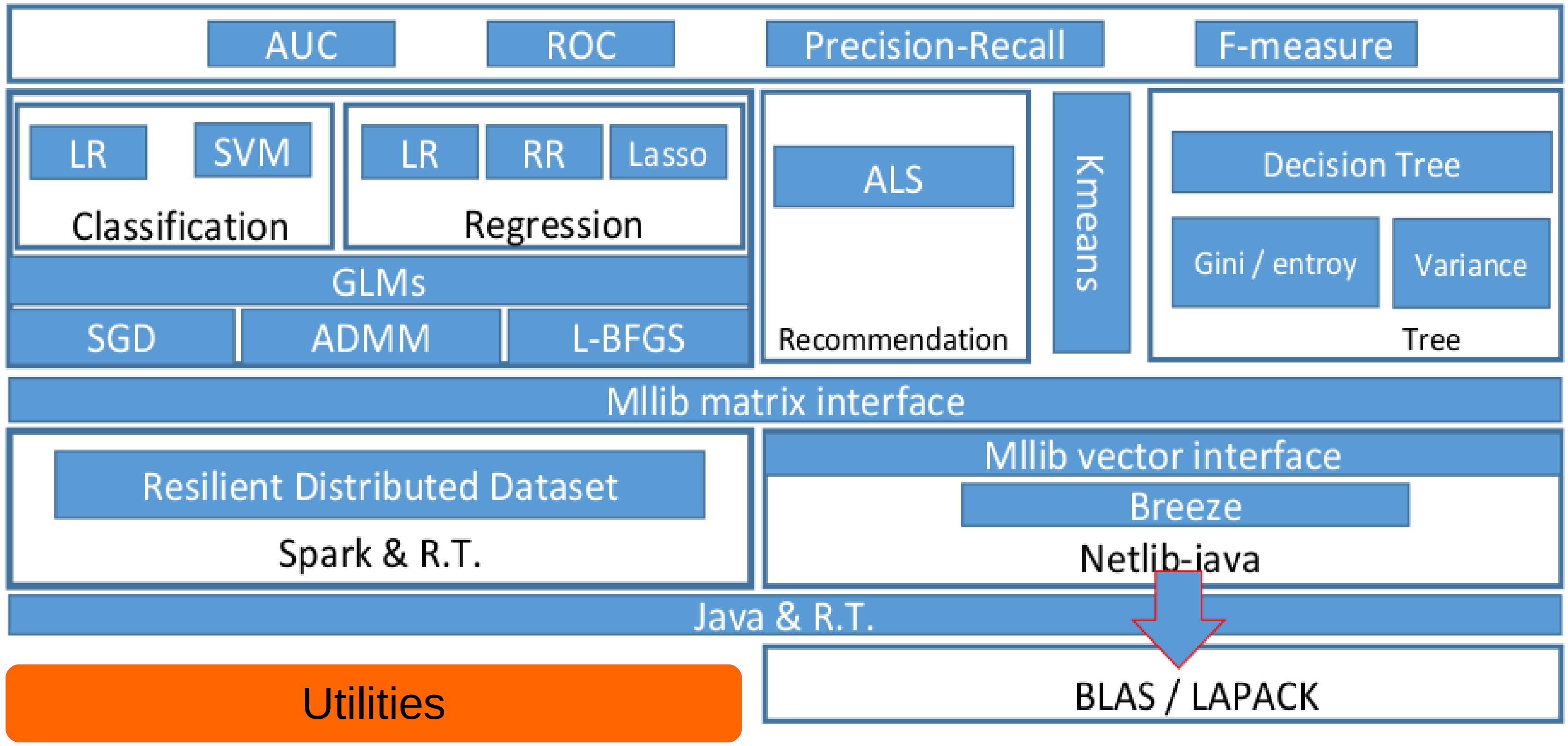
法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

上周回顾



- **MLlib 架构**
- **K-Means**
- **协同过滤**

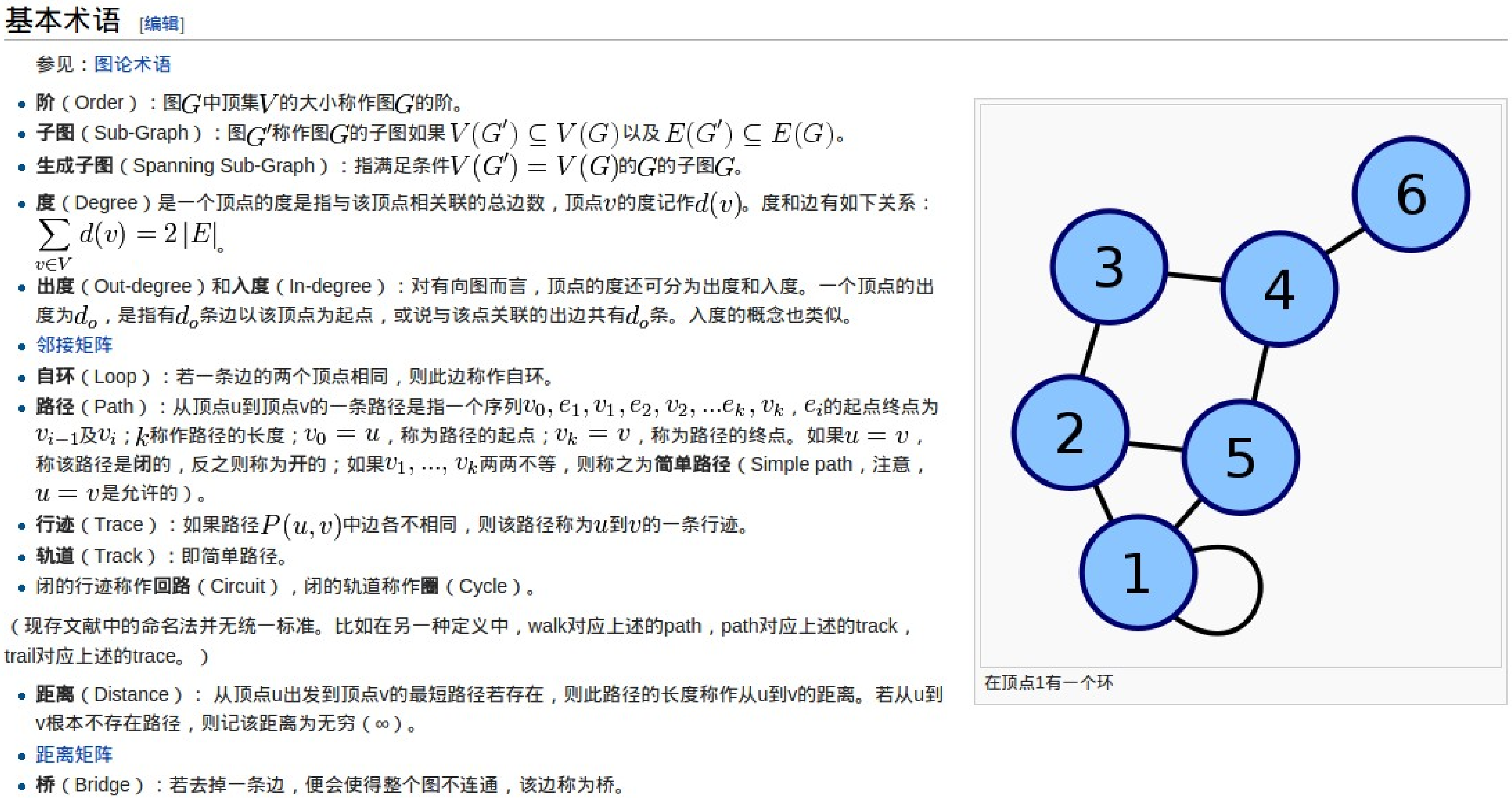
本周内容

- GraphX 简介
  - 图的定义
  - GraphX 图处理
  - GraphX 架构
  - GraphX 操作

- 实例演示
  - 图例演示
  - PageRank 演示

什么是图

- 图是表示物件与物件之间的关系的方法，是图论的基本研究对象。
- 图是由一些小圆点（称为顶点）和连结这些圆点的直线或曲线（称为边）组成的。



基本术语 [编辑]

参见：图论术语

- **阶**（Order）：图G中顶集V的大小称作图G的阶。
- **子图**（Sub-Graph）：图G'称作图G的子图如果$V(G') \subseteq V(G)$以及$E(G') \subseteq E(G)$。
- **生成子图**（Spanning Sub-Graph）：指满足条件$V(G') = V(G)$的G的子图G。
- **度**（Degree）是一个顶点的度是指与该顶点相关联的总边数，顶点v的度记作$d(v)$。度和边有如下关系：$\sum_{v\in V} d(v) = 2\,|E|$。
- **出度**（Out-degree）和**入度**（In-degree）：对有向图而言，顶点的度还可分为出度和入度。一个顶点的出度为d_o，是指有d_o条边以该顶点为起点，或说与该点关联的出边共有d_o条。入度的概念也类似。
- 邻接矩阵
- **自环**（Loop）：若一条边的两个顶点相同，则此边称作自环。
- **路径**（Path）：从顶点u到顶点v的一条路径是指一个序列$v_0, e_1, v_1, e_2, v_2, \ldots e_k, v_k$，$e_i$的起点终点为$v_{i-1}$及$v_i$；$k$称作路径的长度；$v_0 = u$，称为路径的起点；$v_k = v$，称为路径的终点。如果$u = v$，称该路径是闭的，反之则称为开的；如果$v_1, \ldots, v_k$两两不等，则称之为**简单路径**（Simple path，注意，$u = v$是允许的）。
- **行迹**（Trace）：如果路径$P(u, v)$中边各不相同，则该路径称为u到v的一条行迹。
- **轨道**（Track）：即简单路径。
- 闭的行迹称作**回路**（Circuit），闭的轨道称作**圈**（Cycle）。

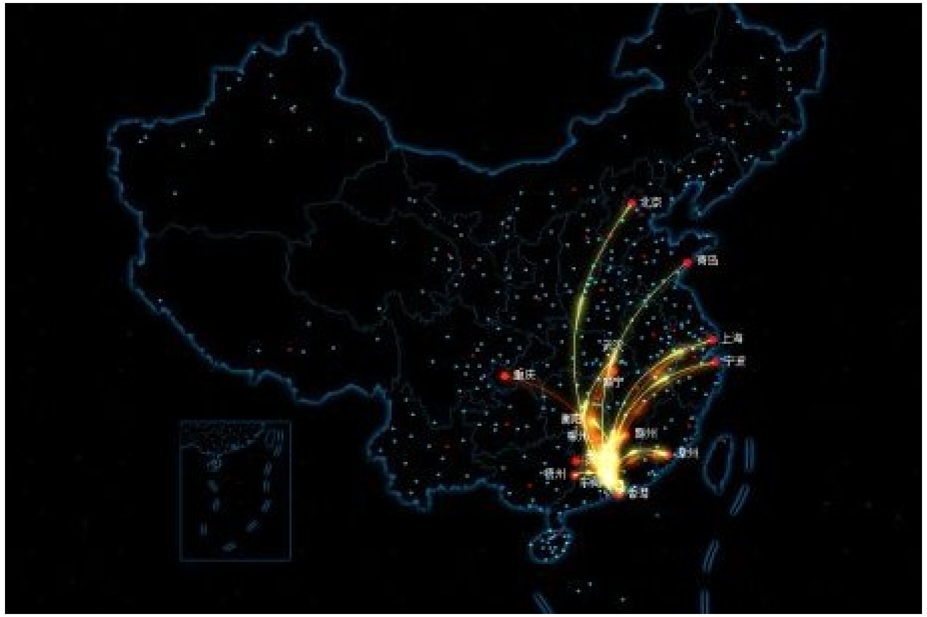
（现存文献中的命名法并无统一标准。比如在另一种定义中，walk对应上述的path，path对应上述的track，trail对应上述的trace。）

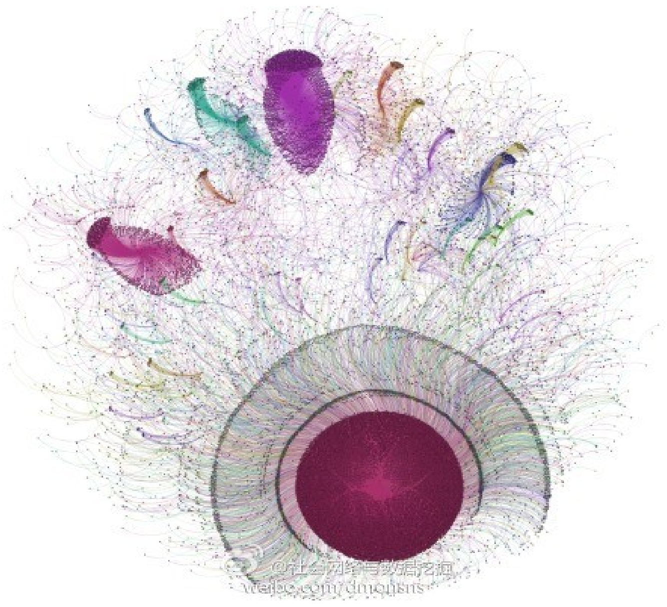
- **距离**（Distance）：从顶点u出发到顶点v的最短路径若存在，则此路径的长度称作从u到v的距离。若从u到v根本不存在路径，则记该距离为无穷（∞）。
- 距离矩阵
- **桥**（Bridge）：若去掉一条边，便会使得整个图不连通，该边称为桥。

在顶点1有一个环

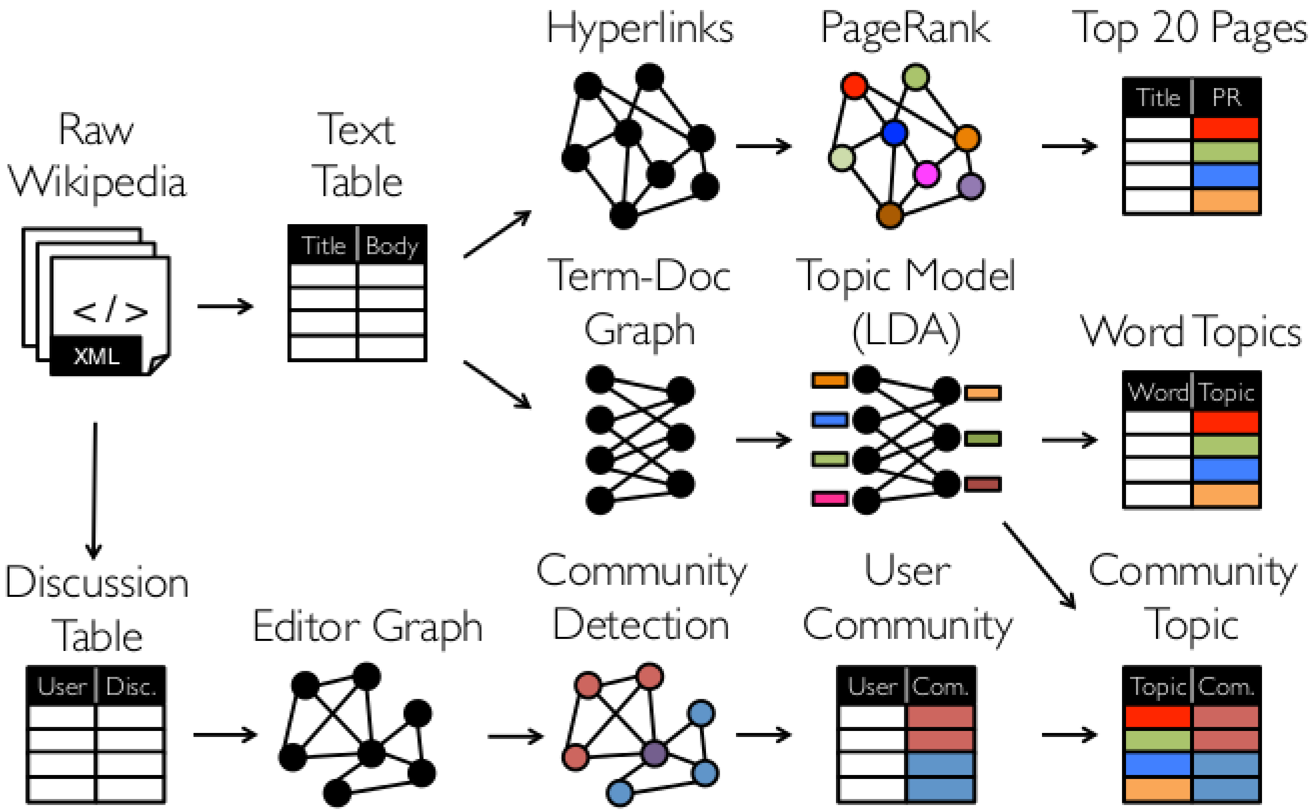
图的应用

- PageRank
- 社交网络
- 网络交易
- 微博
- 交通状况监控
- 监控电商卖家刷评分
- 。。。。

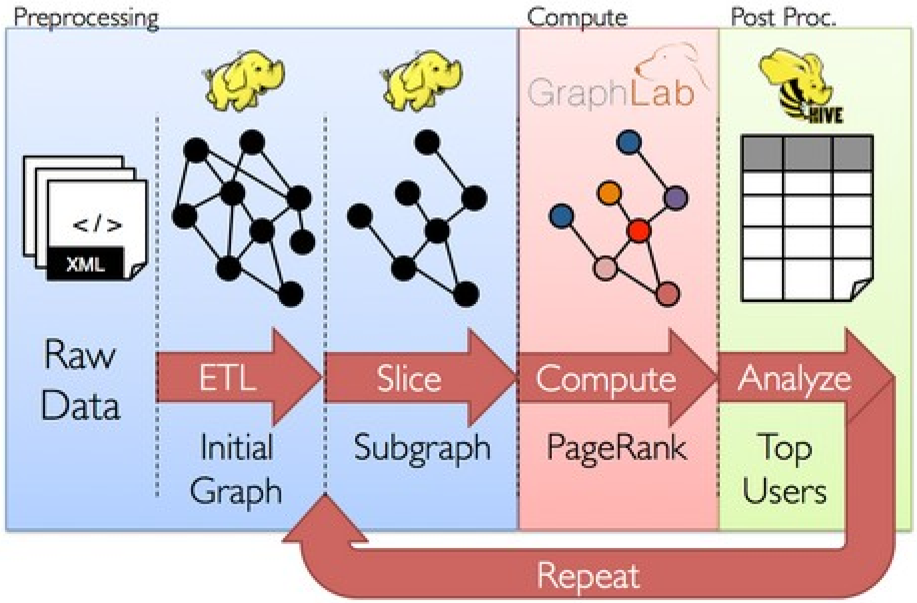




图的处理


Raw Wikipedia
</ >
XML
Text Table
Title
Body
Hyperlinks
PageRank
Top 20 Pages
Title
PR
Term-Doc Graph
Topic Model (LDA)
Word Topics
Word
Topic
Discussion Table
User
Disc.
Editor Graph
Community Detection
User Community
User
Com.
Community Topic
Topic
Com.

图的处理

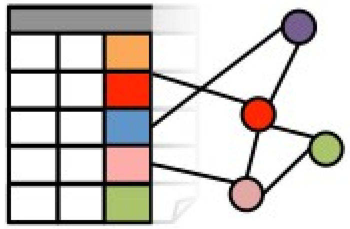

Preprocessing
Compute
Post Proc.
GraphLab
HIVE
</ >
XML
Raw Data
ETL
Initial Graph
Slice
Subgraph
Compute
PageRank
Analyze
Top Users
Repeat

HIVE
Spark
APACHE GIRAPH
GraphLab

GraphX 图处理

New API

Blurs the distinction between Tables and Graphs



New System

Combines Data-Parallel Graph-Parallel Systems

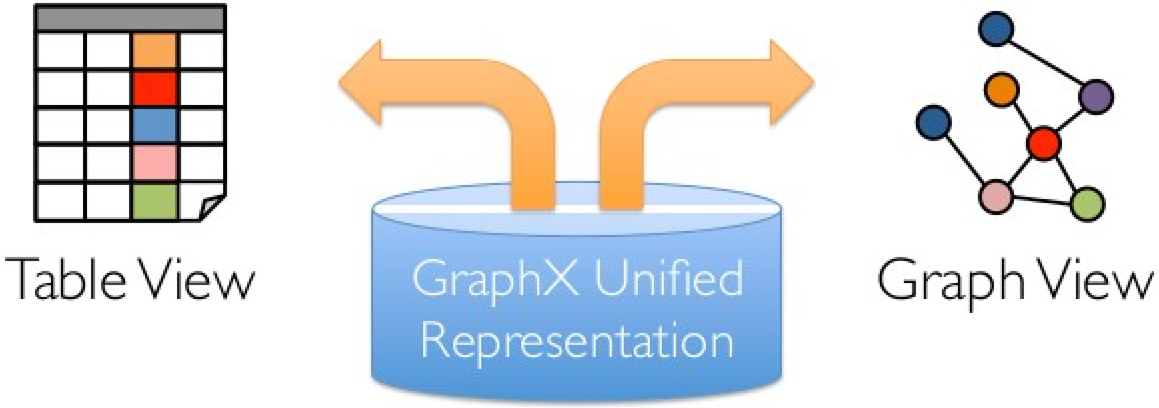


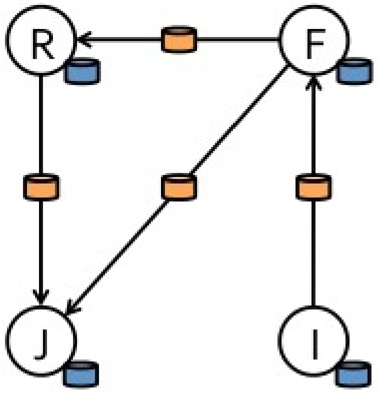
Table View

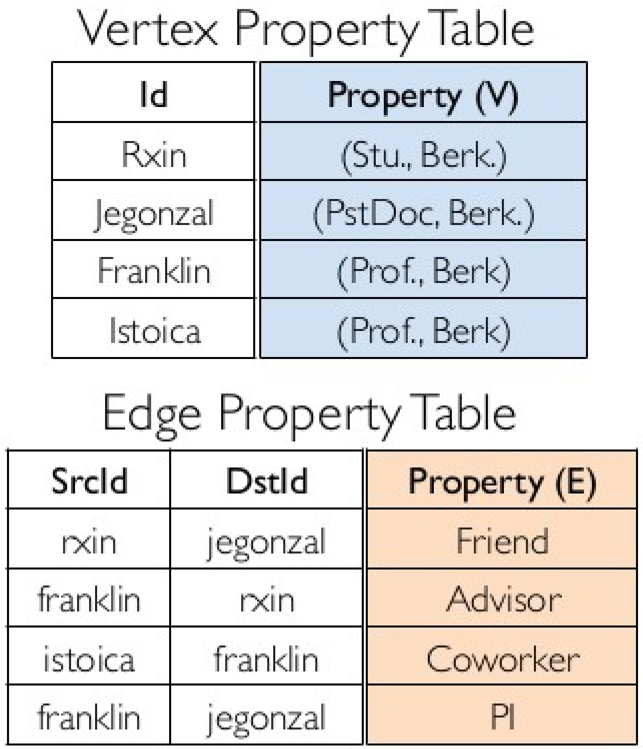
GraphX Unified Representation

Graph View

GraphX 图处理

Property Graph





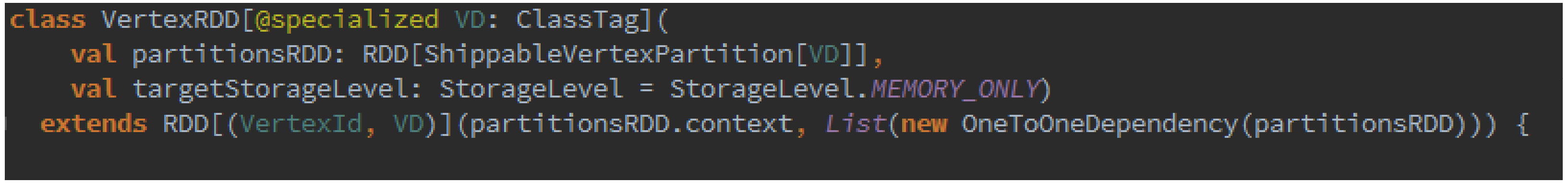
Vertex Property Table

Id	Property (V)
Rxin	(Stu., Berk.)
Jegonzal	(PstDoc, Berk.)
Franklin	(Prof., Berk)
Istoica	(Prof., Berk)

Edge Property Table

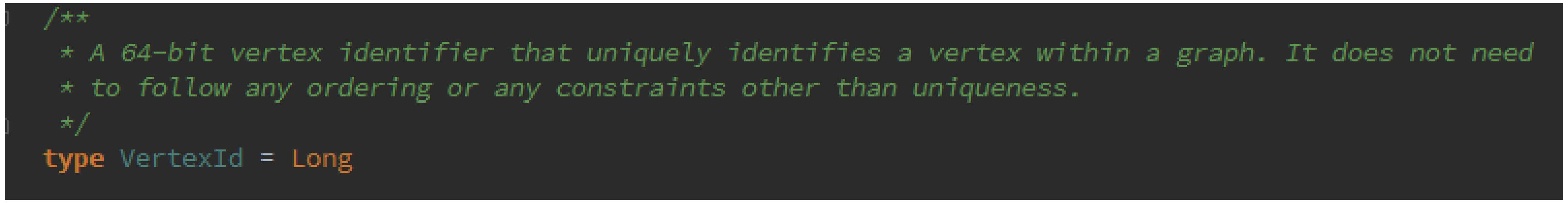
SrcId	DstId	Property (E)
rxin	jegonzal	Friend
franklin	rxin	Advisor
istoica	franklin	Coworker
franklin	jegonzal	PI

GraphX 图处理



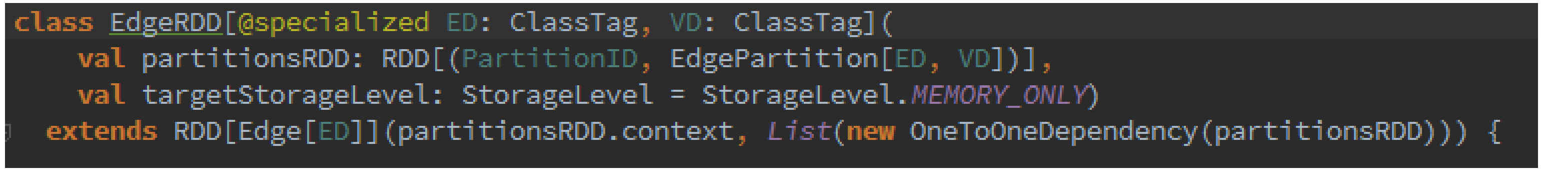
```
class VertexRDD[@specialized VD: ClassTag](
    val partitionsRDD: RDD[ShippableVertexPartition[VD]],
    val targetStorageLevel: StorageLevel = StorageLevel.MEMORY_ONLY)
  extends RDD[(VertexId, VD)](partitionsRDD.context, List(new OneToOneDependency(partitionsRDD))) {
```

顶点



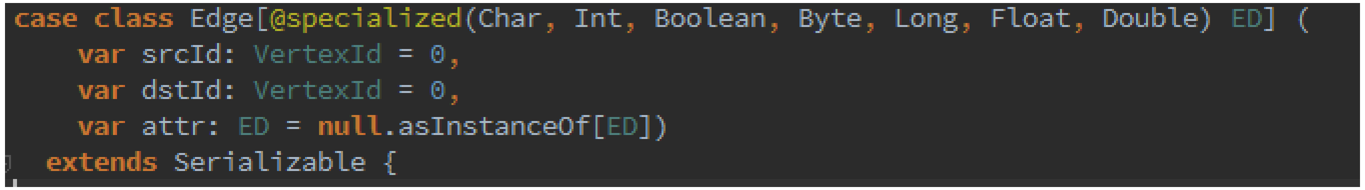
```
  /**
   * A 64-bit vertex identifier that uniquely identifies a vertex within a graph. It does not need
   * to follow any ordering or any constraints other than uniqueness.
   */
  type VertexId = Long
```

GraphX 图处理

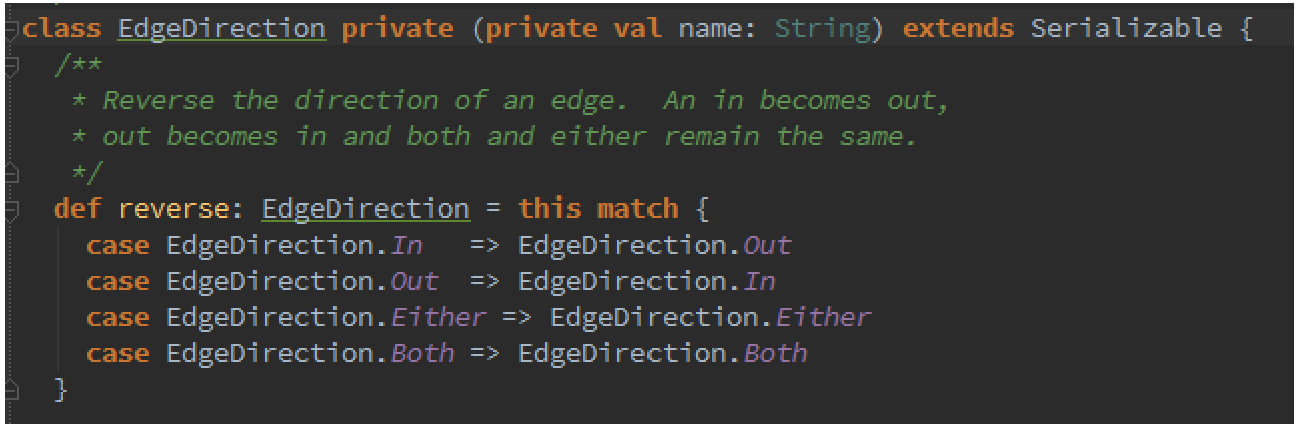


```scala
class EdgeRDD[@specialized ED: ClassTag, VD: ClassTag](
    val partitionsRDD: RDD[(PartitionID, EdgePartition[ED, VD])],
    val targetStorageLevel: StorageLevel = StorageLevel.MEMORY_ONLY)
  extends RDD[Edge[ED]](partitionsRDD.context, List(new OneToOneDependency(partitionsRDD))) {
```

边

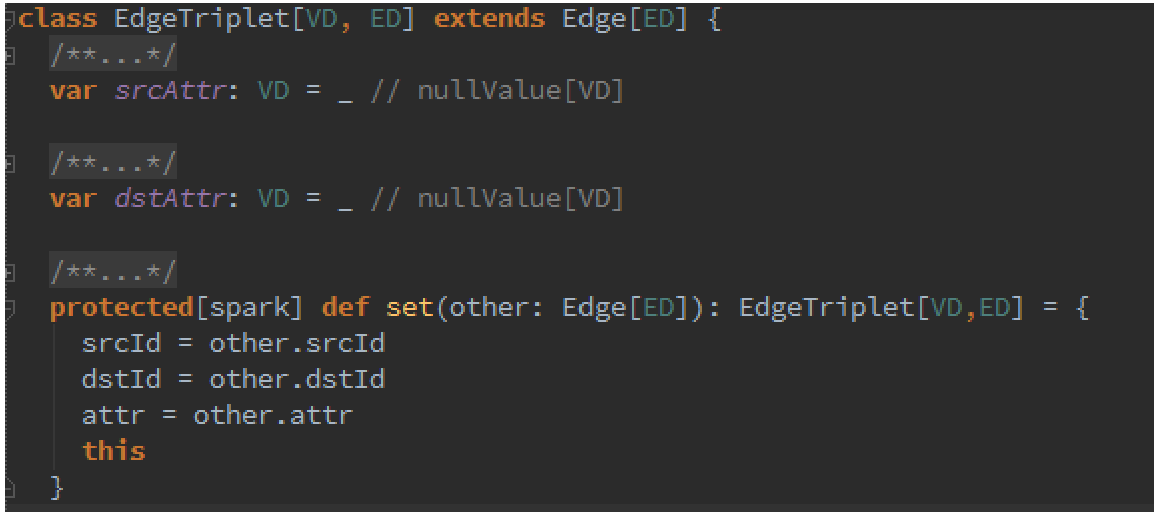


```scala
case class Edge[@specialized(Char, Int, Boolean, Byte, Long, Float, Double) ED] (
    var srcId: VertexId = 0,
    var dstId: VertexId = 0,
    var attr: ED = null.asInstanceOf[ED])
  extends Serializable {
```



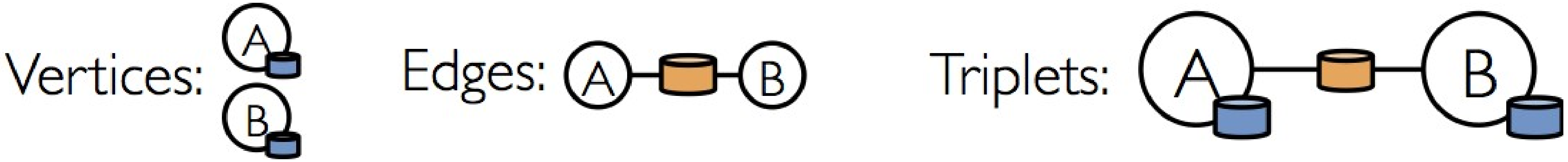
```scala
class EdgeDirection private (private val name: String) extends Serializable {
  /**
   * Reverse the direction of an edge.  An in becomes out,
   * out becomes in and both and either remain the same.
   */
  def reverse: EdgeDirection = this match {
    case EdgeDirection.In   => EdgeDirection.Out
    case EdgeDirection.Out  => EdgeDirection.In
    case EdgeDirection.Either => EdgeDirection.Either
    case EdgeDirection.Both => EdgeDirection.Both
  }
```

GraphX 图处理

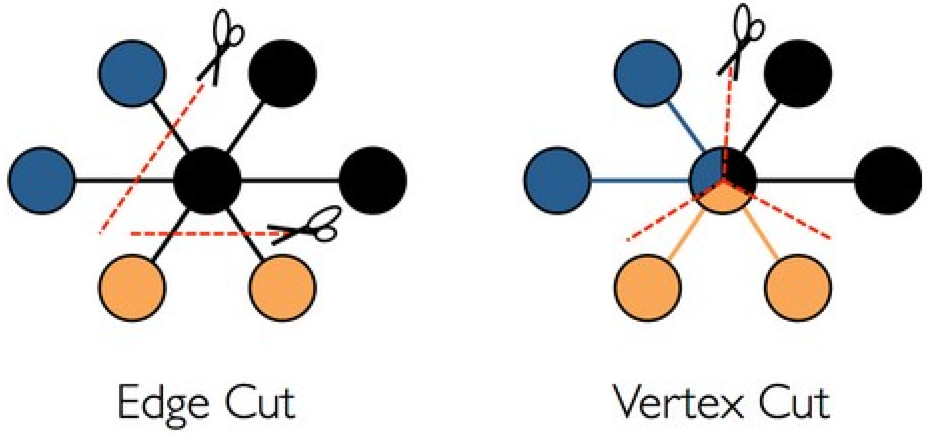


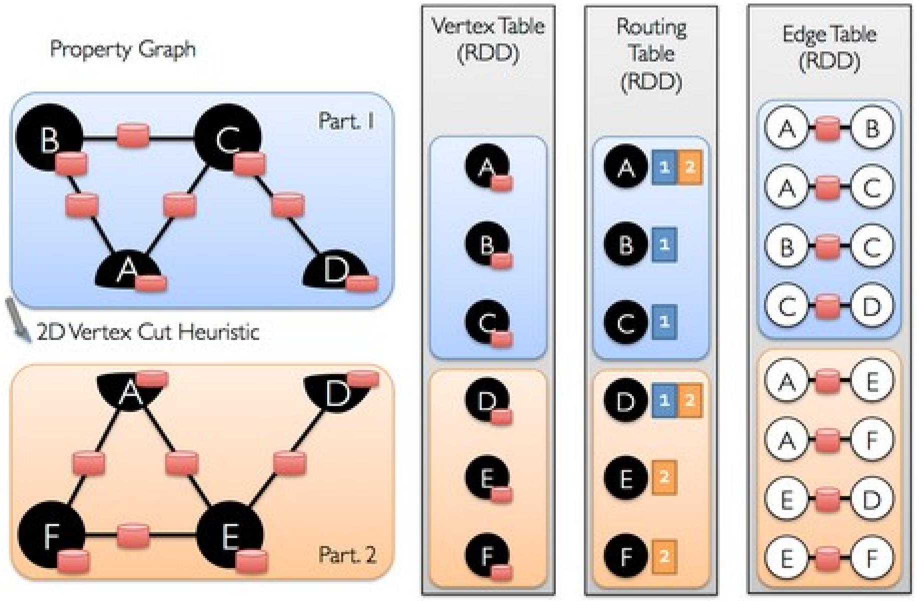
```
class EdgeTriplet[VD, ED] extends Edge[ED] {
  /**...*/
  var srcAttr: VD = _ // nullValue[VD]

  /**...*/
  var dstAttr: VD = _ // nullValue[VD]

  /**...*/
  protected[spark] def set(other: Edge[ED]): EdgeTriplet[VD,ED] = {
    srcId = other.srcId
    dstId = other.dstId
    attr = other.attr
    this
  }
```

Triplet

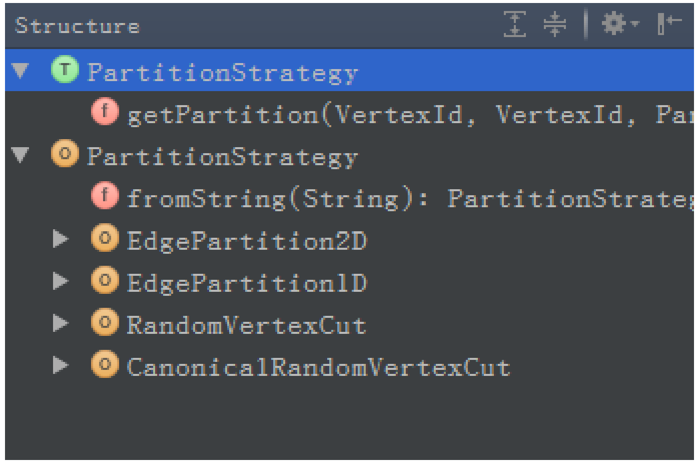


GraphX 图处理



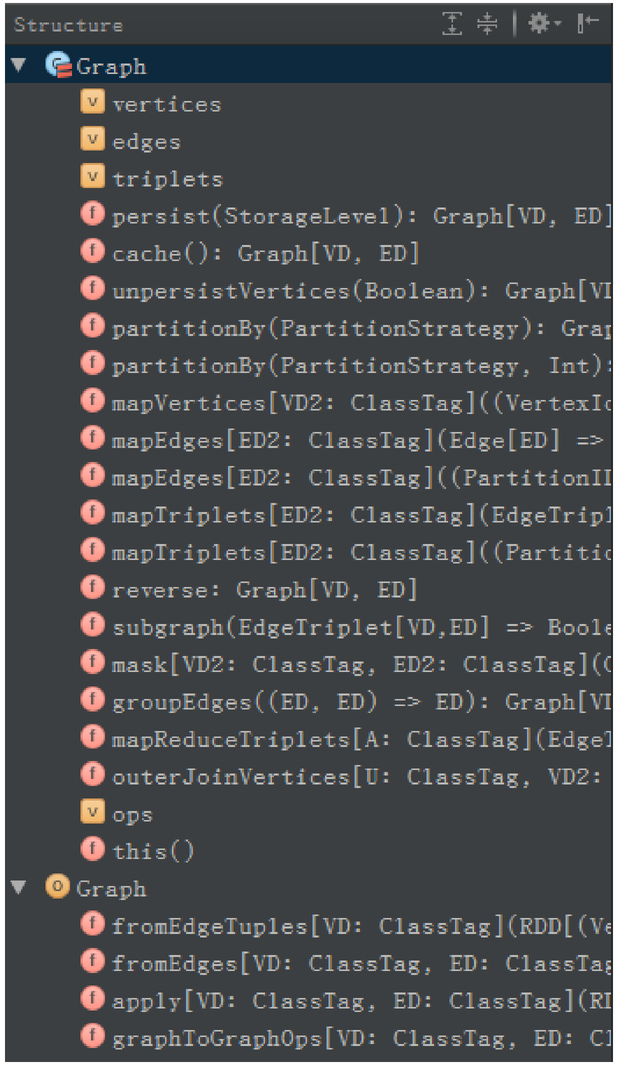


存储

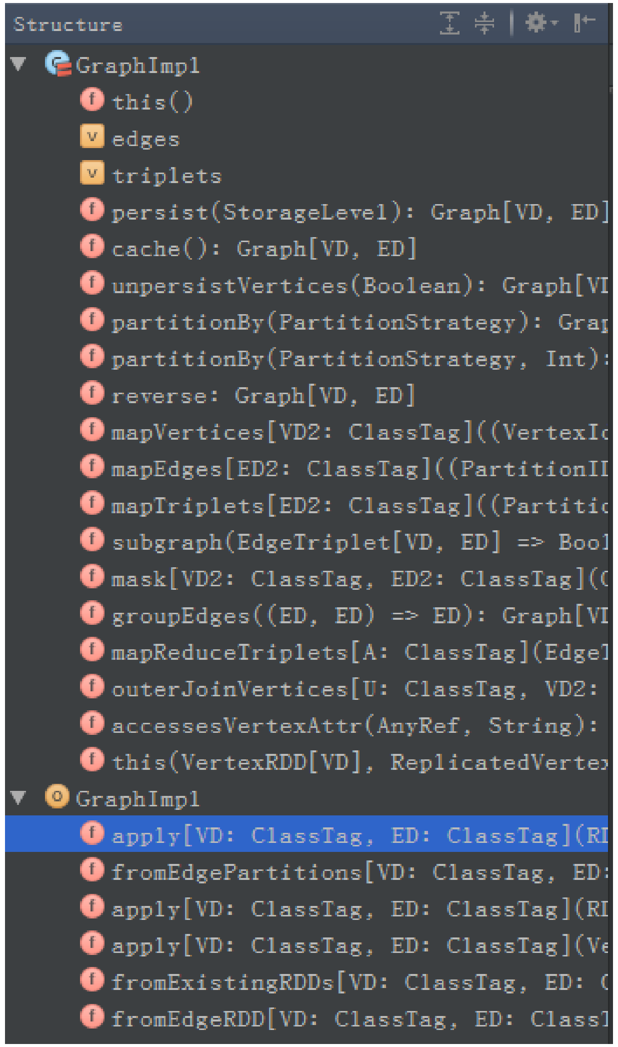


```
Structure
▼ Ⓣ PartitionStrategy
    ⓕ getPartition(VertexId, VertexId, Pa
▼ Ⓞ PartitionStrategy
    ⓕ fromString(String): PartitionStrateg
  ▶ Ⓞ EdgePartition2D
  ▶ Ⓞ EdgePartition1D
  ▶ Ⓞ RandomVertexCut
  ▶ Ⓞ CanonicalRandomVertexCut
```

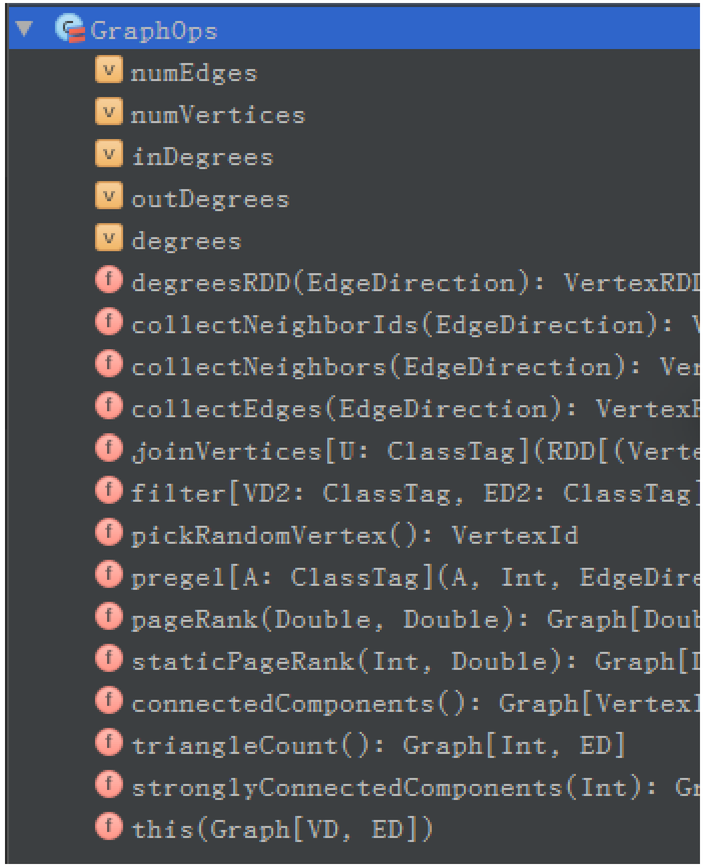
GraphX 图处理



```
Structure
▼ Graph
    vertices
    edges
    triplets
    persist(StorageLevel): Graph[VD, ED]
    cache(): Graph[VD, ED]
    unpersistVertices(Boolean): Graph[VI
    partitionBy(PartitionStrategy): Grap
    partitionBy(PartitionStrategy, Int):
    mapVertices[VD2: ClassTag]((VertexIc
    mapEdges[ED2: ClassTag](Edge[ED] =>
    mapEdges[ED2: ClassTag]((PartitionII
    mapTriplets[ED2: ClassTag](EdgeTripl
    mapTriplets[ED2: ClassTag]((Partitic
    reverse: Graph[VD, ED]
    subgraph(EdgeTriplet[VD,ED] => Boole
    mask[VD2: ClassTag, ED2: ClassTag](C
    groupEdges((ED, ED) => ED): Graph[VI
    mapReduceTriplets[A: ClassTag](Edge1
    outerJoinVertices[U: ClassTag, VD2:
    ops
    this()
▼ Graph
    fromEdgeTuples[VD: ClassTag](RDD[(Ve
    fromEdges[VD: ClassTag, ED: ClassTag
    apply[VD: ClassTag, ED: ClassTag](RI
    graphToGraphOps[VD: ClassTag, ED: Cl
```



```
Structure
▼ GraphImpl
    this()
    edges
    triplets
    persist(StorageLevel): Graph[VD, ED]
    cache(): Graph[VD, ED]
    unpersistVertices(Boolean): Graph[VI
    partitionBy(PartitionStrategy): Grap
    partitionBy(PartitionStrategy, Int):
    reverse: Graph[VD, ED]
    mapVertices[VD2: ClassTag]((VertexIc
    mapEdges[ED2: ClassTag]((PartitionII
    mapTriplets[ED2: ClassTag]((Partitic
    subgraph(EdgeTriplet[VD, ED] => Bool
    mask[VD2: ClassTag, ED2: ClassTag](C
    groupEdges((ED, ED) => ED): Graph[VI
    mapReduceTriplets[A: ClassTag](Edge1
    outerJoinVertices[U: ClassTag, VD2:
    accessesVertexAttr(AnyRef, String):
    this(VertexRDD[VD], ReplicatedVertex
▼ GraphImpl
    apply[VD: ClassTag, ED: ClassTag](RI
    fromEdgePartitions[VD: ClassTag, ED:
    apply[VD: ClassTag, ED: ClassTag](RI
    apply[VD: ClassTag, ED: ClassTag](Ve
    fromExistingRDDs[VD: ClassTag, ED: (
    fromEdgeRDD[VD: ClassTag, ED: Class1
```

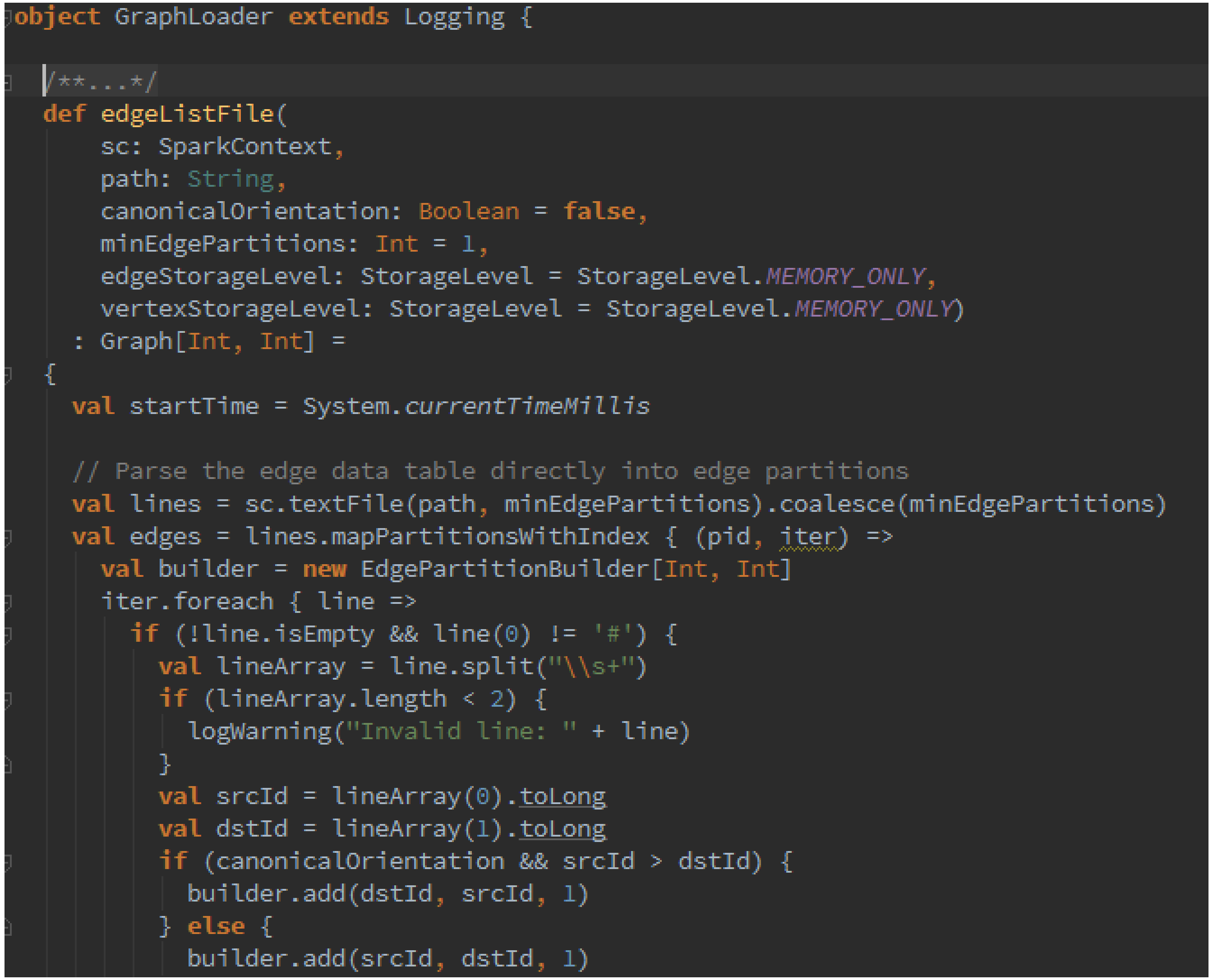


```
▼ GraphOps
    numEdges
    numVertices
    inDegrees
    outDegrees
    degrees
    degreesRDD(EdgeDirection): VertexRDI
    collectNeighborIds(EdgeDirection): V
    collectNeighbors(EdgeDirection): Ver
    collectEdges(EdgeDirection): VertexF
    joinVertices[U: ClassTag](RDD[(Verte
    filter[VD2: ClassTag, ED2: ClassTag]
    pickRandomVertex(): VertexId
    pregel[A: ClassTag](A, Int, EdgeDire
    pageRank(Double, Double): Graph[Douk
    staticPageRank(Int, Double): Graph[I
    connectedComponents(): Graph[VertexI
    triangleCount(): Graph[Int, ED]
    stronglyConnectedComponents(Int): Gr
    this(Graph[VD, ED])
```

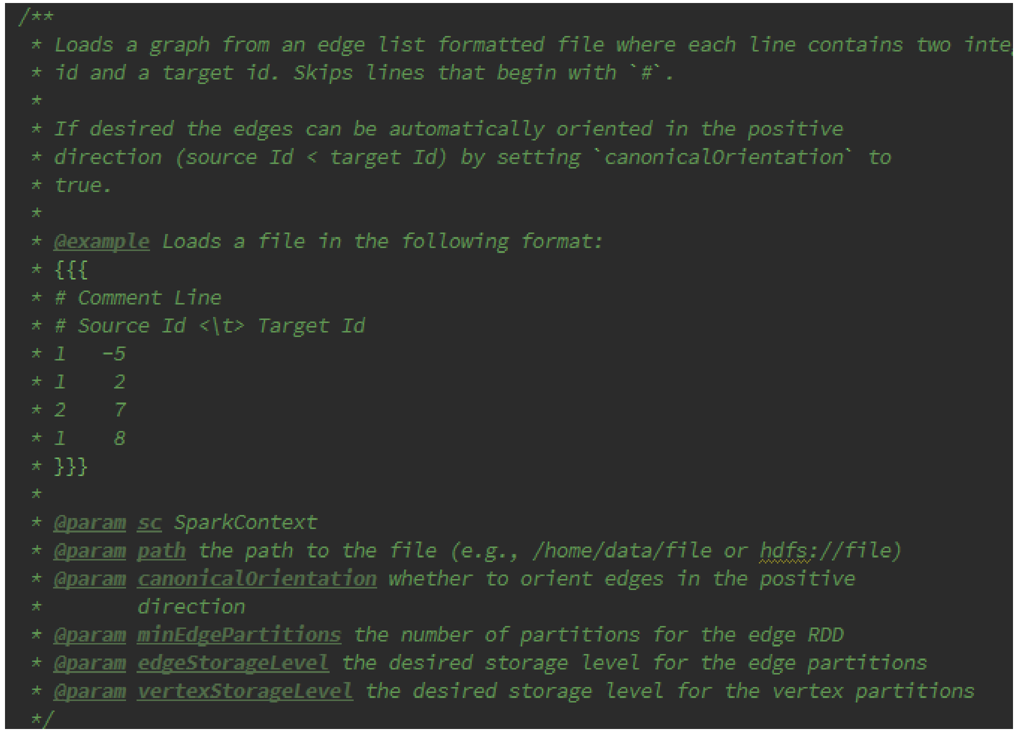
graph

DATAGURU 专业数据分析社区

GraphX 图处理



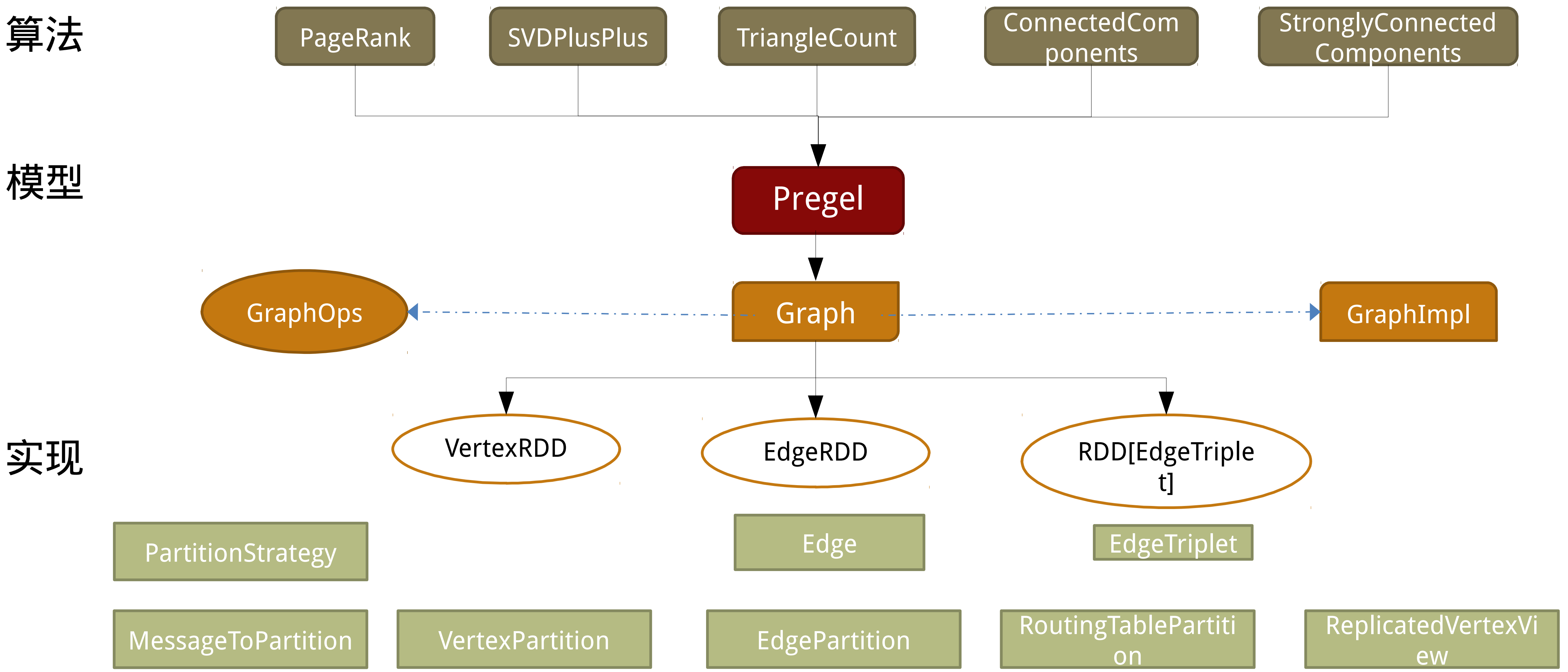
```
object GraphLoader extends Logging {

  /**...*/
  def edgeListFile(
      sc: SparkContext,
      path: String,
      canonicalOrientation: Boolean = false,
      minEdgePartitions: Int = 1,
      edgeStorageLevel: StorageLevel = StorageLevel.MEMORY_ONLY,
      vertexStorageLevel: StorageLevel = StorageLevel.MEMORY_ONLY)
    : Graph[Int, Int] =
  {
    val startTime = System.currentTimeMillis

    // Parse the edge data table directly into edge partitions
    val lines = sc.textFile(path, minEdgePartitions).coalesce(minEdgePartitions)
    val edges = lines.mapPartitionsWithIndex { (pid, iter) =>
      val builder = new EdgePartitionBuilder[Int, Int]
      iter.foreach { line =>
        if (!line.isEmpty && line(0) != '#') {
          val lineArray = line.split("\\s+")
          if (lineArray.length < 2) {
            logWarning("Invalid line: " + line)
          }
          val srcId = lineArray(0).toLong
          val dstId = lineArray(1).toLong
          if (canonicalOrientation && srcId > dstId) {
            builder.add(dstId, srcId, 1)
          } else {
            builder.add(srcId, dstId, 1)
```



```
/**
 * Loads a graph from an edge list formatted file where each line contains two inte
 * id and a target id. Skips lines that begin with `#`.
 *
 * If desired the edges can be automatically oriented in the positive
 * direction (source Id < target Id) by setting `canonicalOrientation` to
 * true.
 *
 * @example Loads a file in the following format:
 * {{{
 * # Comment Line
 * # Source Id <\t> Target Id
 * 1   -5
 * 1    2
 * 2    7
 * 1    8
 * }}}
 *
 * @param sc SparkContext
 * @param path the path to the file (e.g., /home/data/file or hdfs://file)
 * @param canonicalOrientation whether to orient edges in the positive
 *        direction
 * @param minEdgePartitions the number of partitions for the edge RDD
 * @param edgeStorageLevel the desired storage level for the edge partitions
 * @param vertexStorageLevel the desired storage level for the vertex partitions
 */
```

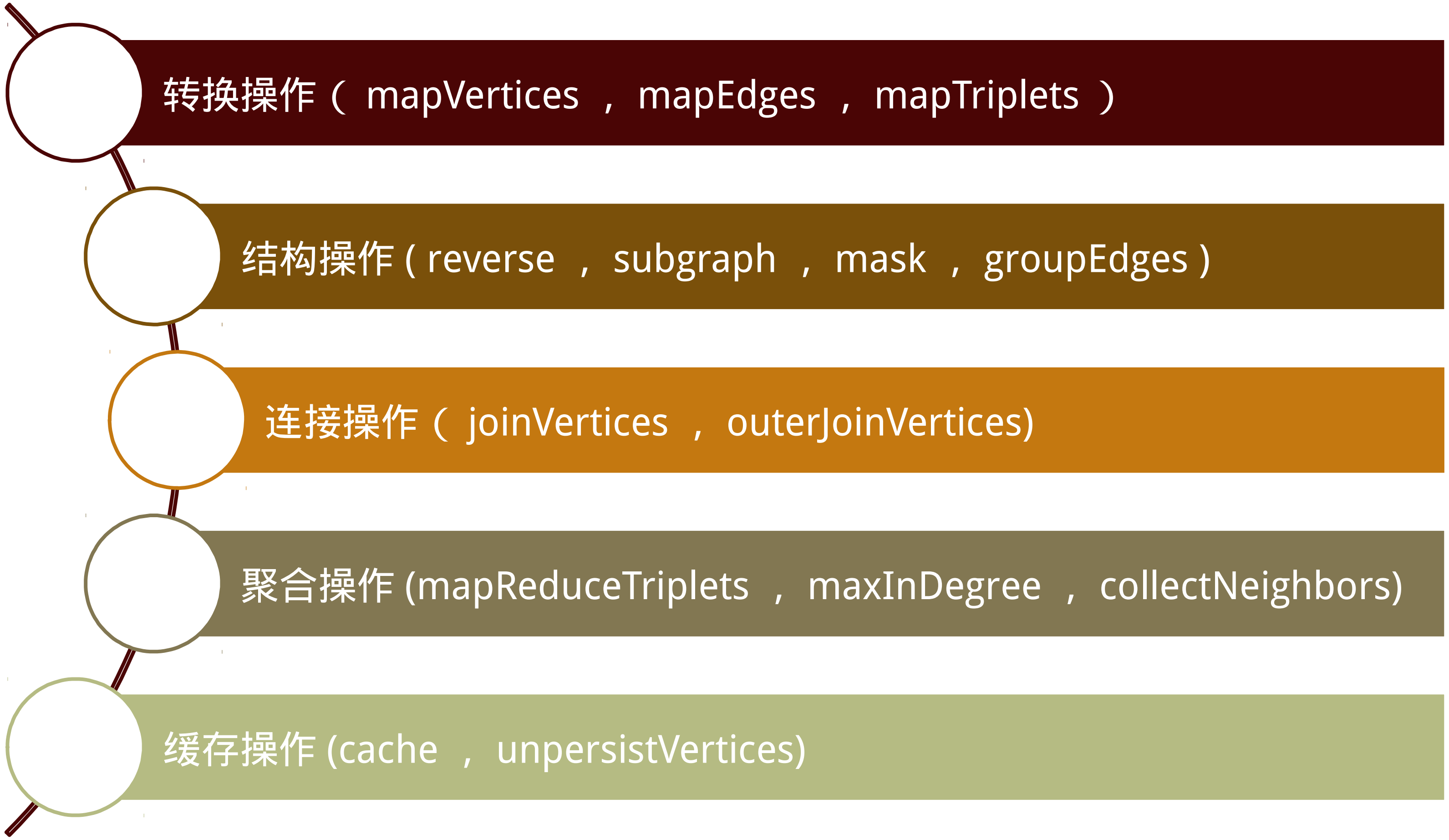
graph

GraphX 架构



算法
PageRank
SVDPlusPlus
TriangleCount
ConnectedComponents
StronglyConnectedComponents
模型
Pregel
GraphOps
Graph
GraphImpl
实现
VertexRDD
EdgeRDD
RDD[EdgeTriplet]
PartitionStrategy
Edge
EdgeTriplet
MessageToPartition
VertexPartition
EdgePartition
RoutingTablePartition
ReplicatedVertexView

GraphX 操作



转换操作（ mapVertices ， mapEdges ， mapTriplets ）

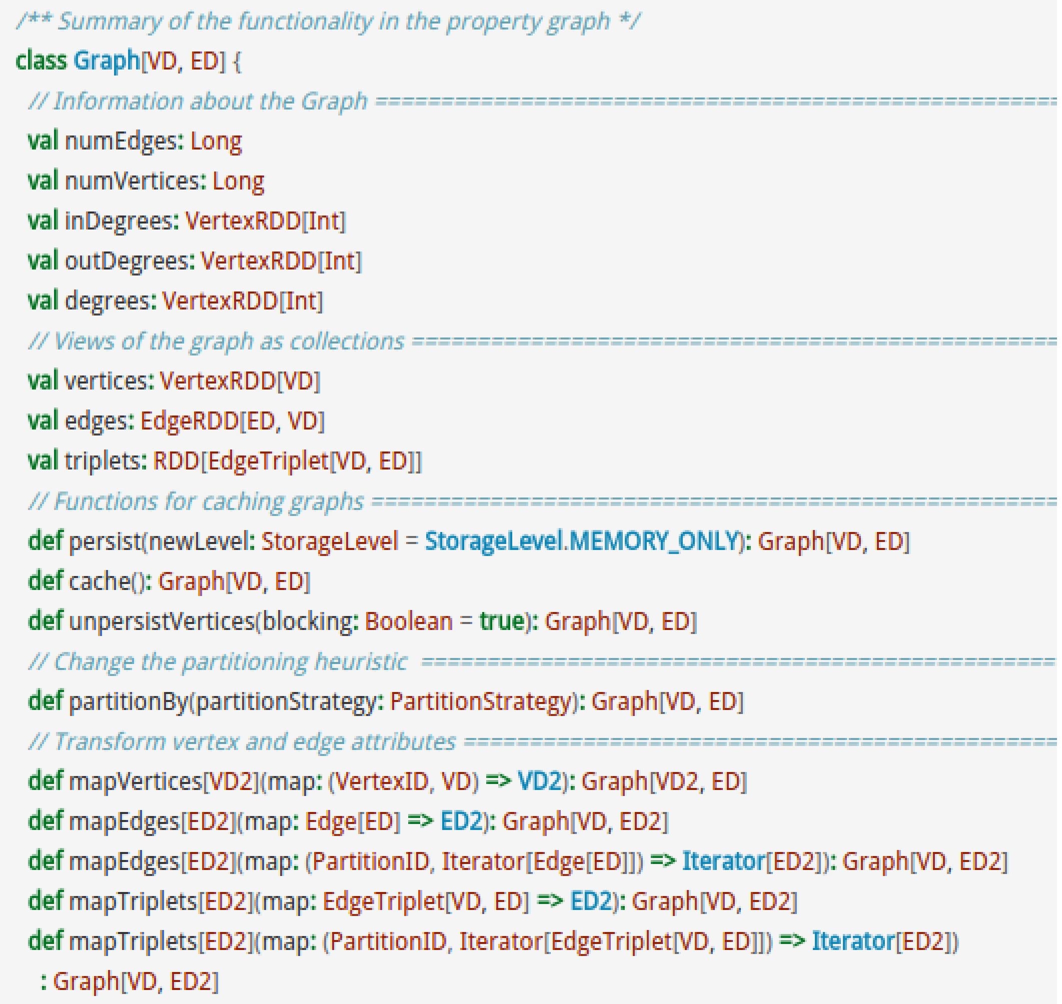
结构操作 (reverse ， subgraph ， mask ， groupEdges)

连接操作（ joinVertices ， outerJoinVertices)

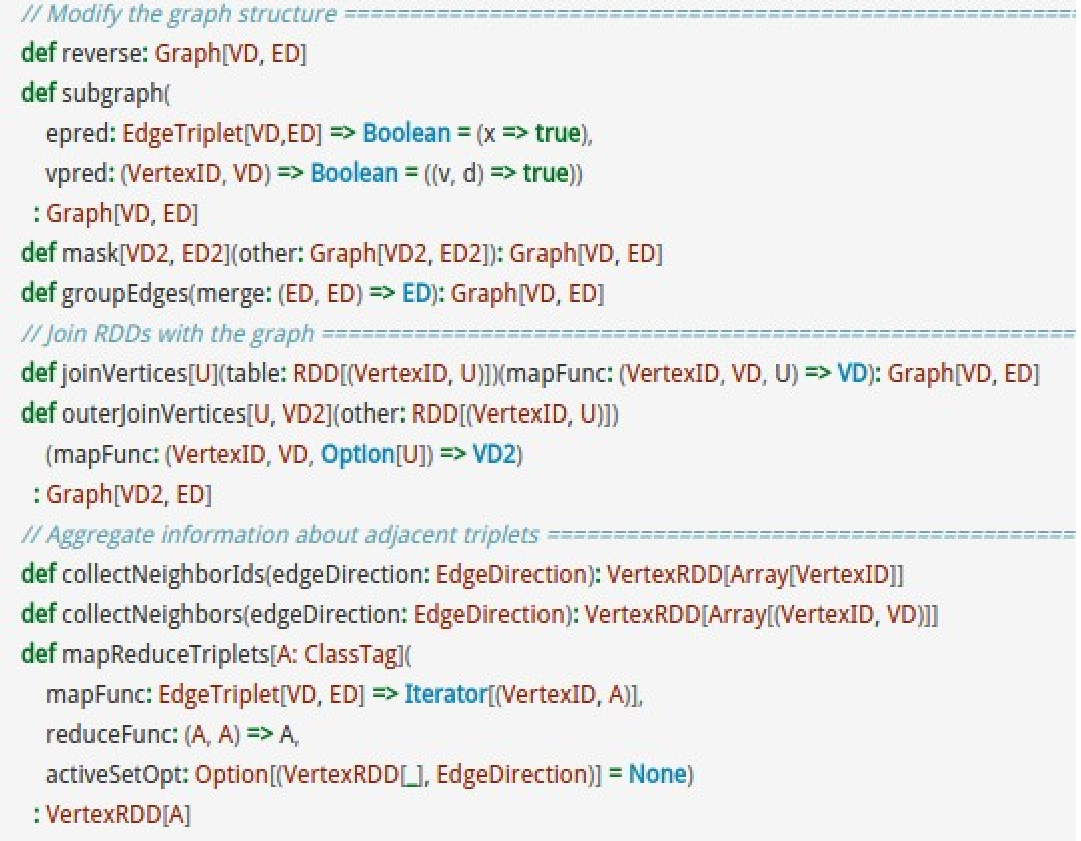
聚合操作 (mapReduceTriplets ， maxInDegree ， collectNeighbors)

缓存操作 (cache ， unpersistVertices)

GraphX 操作



```
/** Summary of the functionality in the property graph */
class Graph[VD, ED] {
  // Information about the Graph ===================================================
  val numEdges: Long
  val numVertices: Long
  val inDegrees: VertexRDD[Int]
  val outDegrees: VertexRDD[Int]
  val degrees: VertexRDD[Int]
  // Views of the graph as collections =============================================
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED, VD]
  val triplets: RDD[EdgeTriplet[VD, ED]]
  // Functions for caching graphs ==================================================
  def persist(newLevel: StorageLevel = StorageLevel.MEMORY_ONLY): Graph[VD, ED]
  def cache(): Graph[VD, ED]
  def unpersistVertices(blocking: Boolean = true): Graph[VD, ED]
  // Change the partitioning heuristic  ============================================
  def partitionBy(partitionStrategy: PartitionStrategy): Graph[VD, ED]
  // Transform vertex and edge attributes ==========================================
  def mapVertices[VD2](map: (VertexID, VD) => VD2): Graph[VD2, ED]
  def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]
  def mapEdges[ED2](map: (PartitionID, Iterator[Edge[ED]]) => Iterator[ED2]): Graph[VD, ED2]
  def mapTriplets[ED2](map: EdgeTriplet[VD, ED] => ED2): Graph[VD, ED2]
  def mapTriplets[ED2](map: (PartitionID, Iterator[EdgeTriplet[VD, ED]]) => Iterator[ED2])
    : Graph[VD, ED2]
```



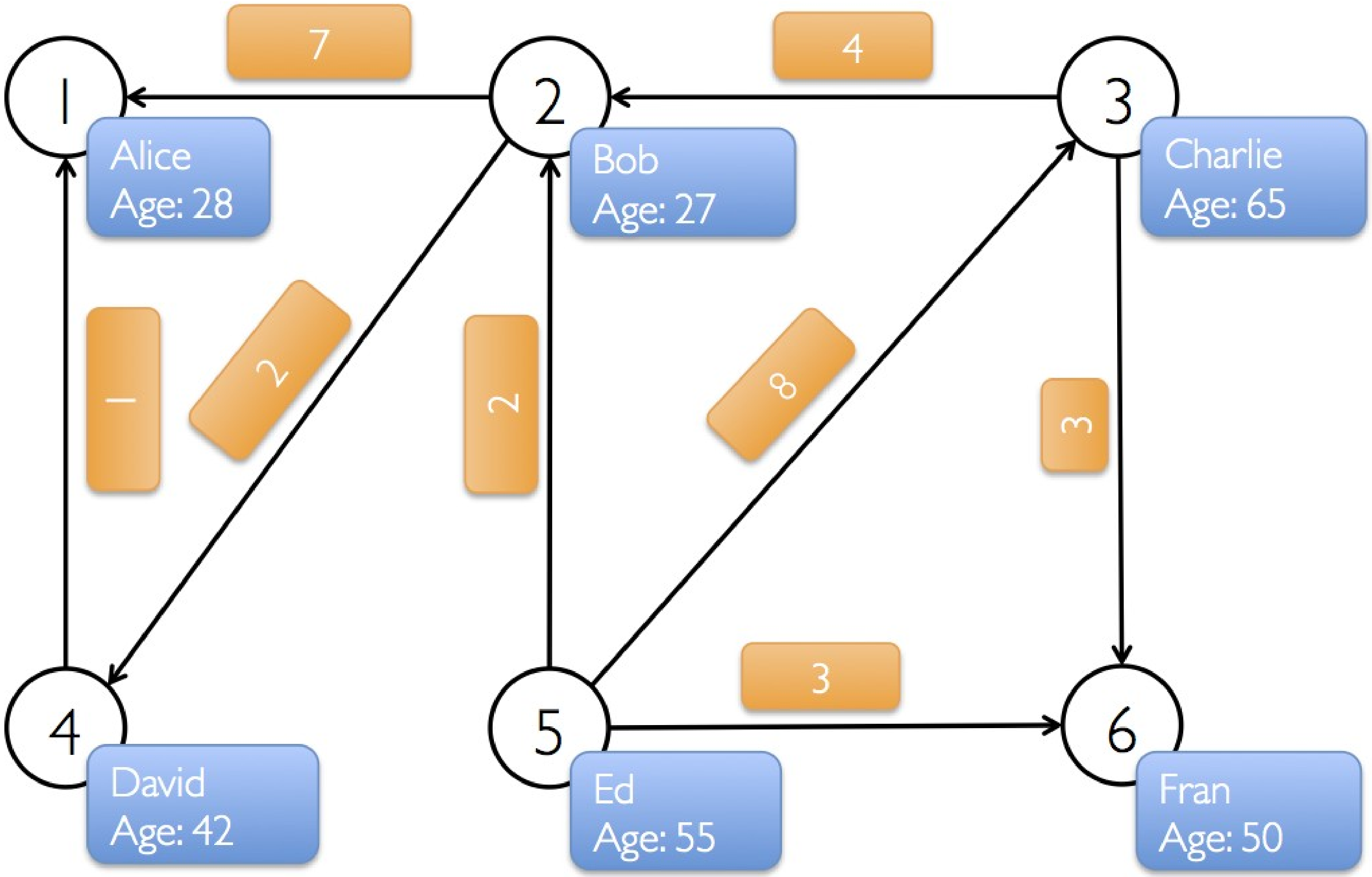
```
// Modify the graph structure ====================================================
def reverse: Graph[VD, ED]
def subgraph(
    epred: EdgeTriplet[VD,ED] => Boolean = (x => true),
    vpred: (VertexID, VD) => Boolean = ((v, d) => true))
  : Graph[VD, ED]
def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED]
def groupEdges(merge: (ED, ED) => ED): Graph[VD, ED]
// Join RDDs with the graph ======================================================
def joinVertices[U](table: RDD[(VertexID, U)])(mapFunc: (VertexID, VD, U) => VD): Graph[VD, ED]
def outerJoinVertices[U, VD2](other: RDD[(VertexID, U)])
    (mapFunc: (VertexID, VD, Option[U]) => VD2)
  : Graph[VD2, ED]
// Aggregate information about adjacent triplets =================================
def collectNeighborIds(edgeDirection: EdgeDirection): VertexRDD[Array[VertexID]]
def collectNeighbors(edgeDirection: EdgeDirection): VertexRDD[Array[(VertexID, VD)]]
def mapReduceTriplets[A: ClassTag](
    mapFunc: EdgeTriplet[VD, ED] => Iterator[(VertexID, A)],
    reduceFunc: (A, A) => A,
    activeSetOpt: Option[(VertexRDD[_], EdgeDirection)] = None)
  : VertexRDD[A]
```

本周内容

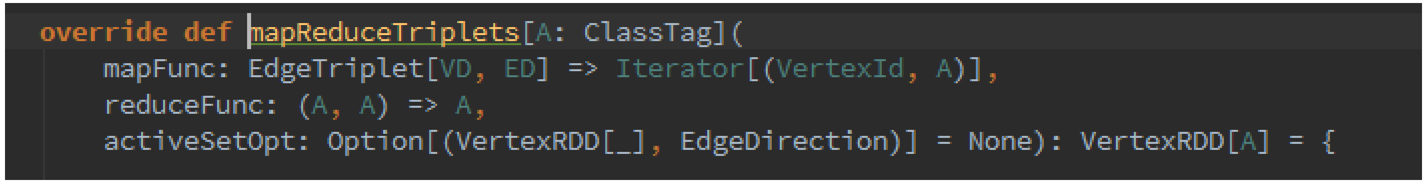
- GraphX 简介
  - 图的定义
  - GraphX 图处理
  - GraphX 架构
  - GraphX 操作

- 实例演示
  - 图例演示
  - PageRank 演示

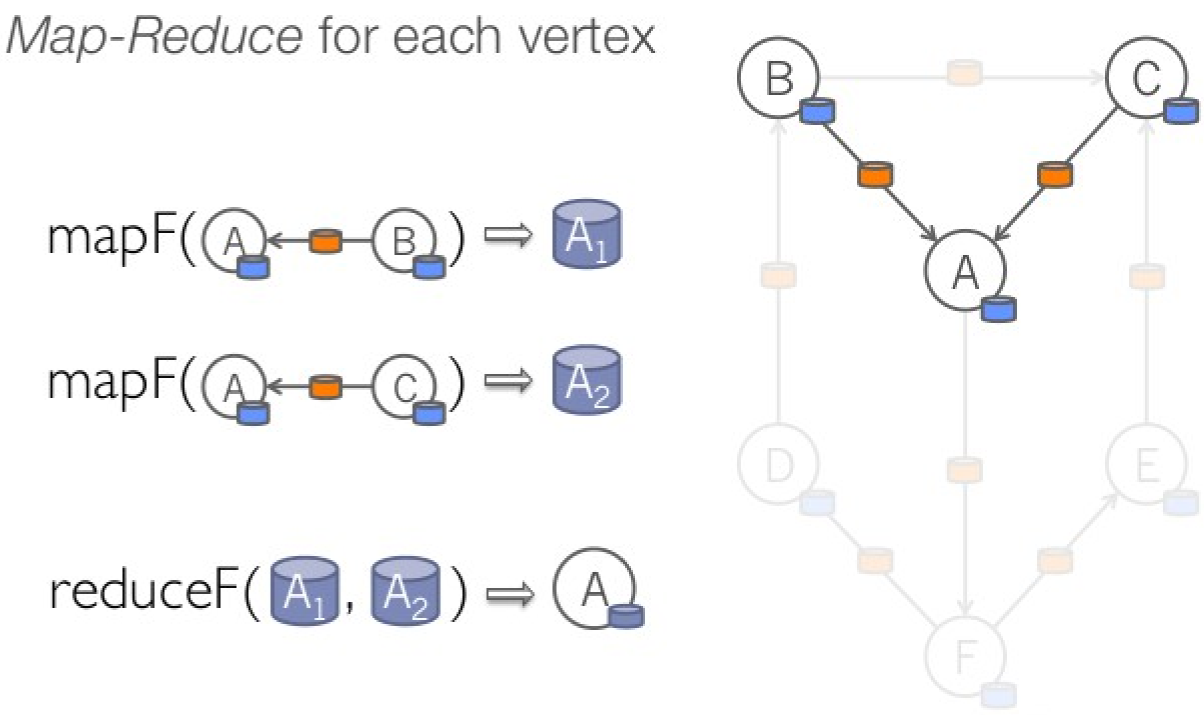
图例演示


7
4
1
Alice
Age: 28
2
Bob
Age: 27
3
Charlie
Age: 65
1
2
2
8
3
3
4
David
Age: 42
5
Ed
Age: 55
6
Fran
Age: 50

图例演示

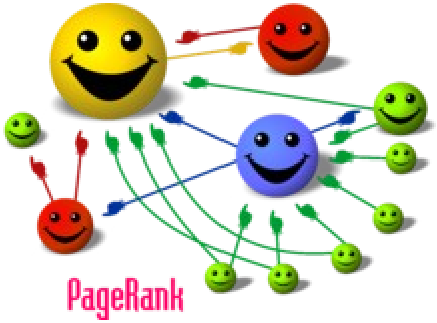


```
override def mapReduceTriplets[A: ClassTag](
    mapFunc: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
    reduceFunc: (A, A) => A,
    activeSetOpt: Option[(VertexRDD[_], EdgeDirection)] = None): VertexRDD[A] = {
```



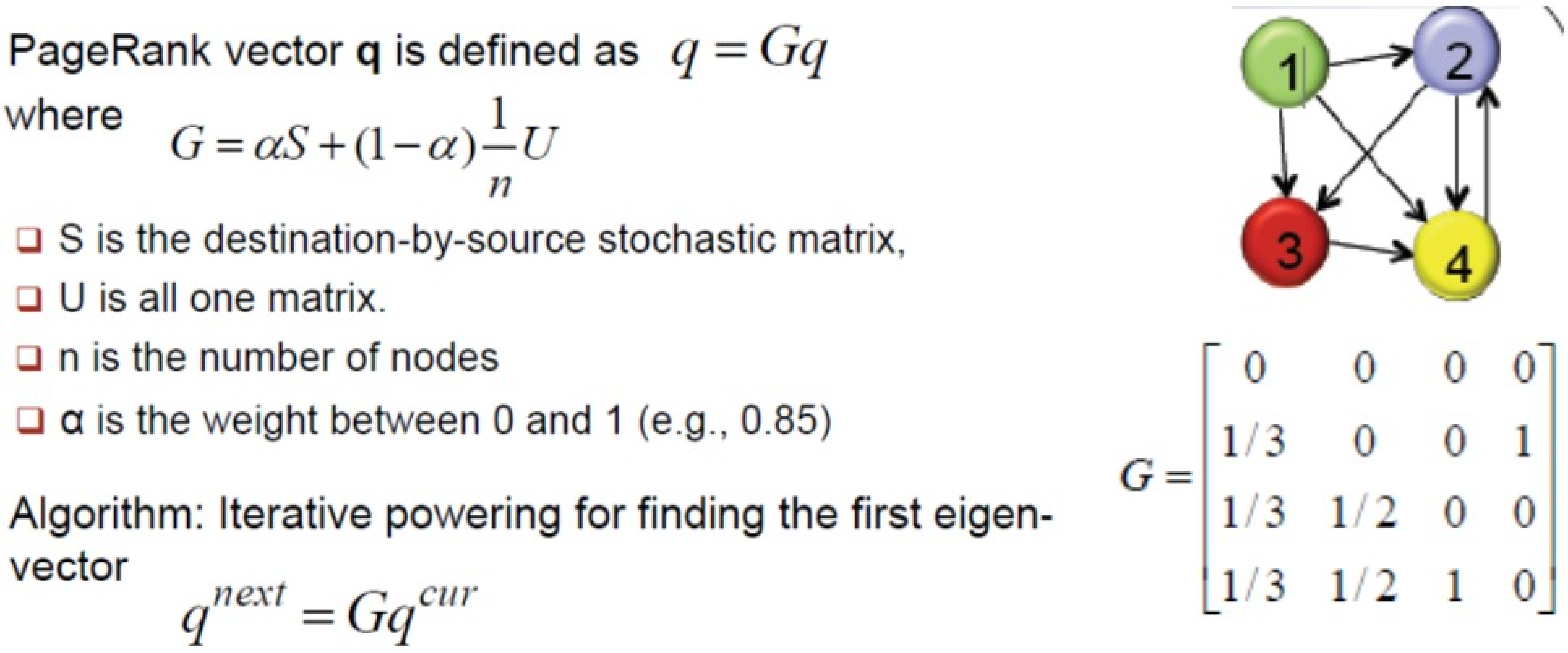
PageRank

- PageRank, 即网页排名，又称网页级别、 Google 左侧排名或佩奇排名。
  - 是 Google 创始人拉里 · 佩奇和谢尔盖 · 布林于 1997 年构建早期的搜索系统原型时提出的链接分析算法。
  - 目前很多重要的链接分析算法都是在 PageRank 算法基础上衍生出来的。
- PageRank 是 Google 用于用来标识网页的等级 / 重要性的一种方法，是 Google 用来衡量一个网站的好坏的唯一标准。
- 在揉合了诸如 Title 标识和 Keywords 标识等所有其它因素之后， Google 通过 PageRank 来调整结果，使那些更具“等级 / 重要性”的网页在搜索结果中令网站排名获得提升，从而提高搜索结果的相关性和质量。
- 其级别从 0 到 10 级， 10 级为满分。 PR 值越高说明该网页越受欢迎（越重要）。



PageRank

这是Google最核心的算法，用于给每个网页价值评分，是Google“在垃圾中找黄金”的关键算法，这个算法成就了今天的Google



PageRank vector **q** is defined as $q = Gq$

where $G = \alpha S + (1-\alpha)\frac{1}{n}U$

- ❑ S is the destination-by-source stochastic matrix,
- ❑ U is all one matrix.
- ❑ n is the number of nodes
- ❑ α is the weight between 0 and 1 (e.g., 0.85)

Algorithm: Iterative powering for finding the first eigen-vector

$$q^{next} = Gq^{cur}$$

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1 \\ 1/3 & 1/2 & 0 & 0 \\ 1/3 & 1/2 & 1 & 0 \end{bmatrix}$$

课程回顾

- GraphX 构成
- GraphX 操作

- 下周运维和优化
  - 运维（参数配置、 HistoryServer 、 Jobserver 、 ganglia 集成）
  - 优化（序化、压缩。。。）

See You Next

Thanks

FAQ 时间