Gelly:Graph Algorithms

在 Gelly 中作为图算法，`Graph` API 和 顶层算法集成的逻辑模块都在 `org.apache.flink.graph.asm` 中。这些算法可用通过配置参数进行优化和调整，并且当用一组相似的配置对相同的输入进行处理时，提供隐式的运行时复用。.

Back to top

算法	描述

degree.annotate.directed.
VertexInDegree
	用入边 (in-degree) 标注一个有向图的点.
	`DataSet<Vertex<K, LongValue>> inDegree = graph`
	`  .run(new VertexInDegree()`
	`  .setIncludeZeroDegreeVertices(true));`
	可选配置:
	• **setIncludeZeroDegreeVertices**: 默认情况下为了自由度的计算，只有边集 (edge set) 需要被处理；当该参数被设置时，对点集 (vertex set) 会进行一个额外的 join 操作来输出入边数 (in-degree) 为 0 的点
	• **setParallelism**: 指定算子的并行度

degree.annotate.directed.
VertexOutDegree
	用出边 (out-degree) 标注一个有向图的点.
	`DataSet<Vertex<K, LongValue>> outDegree = graph`
	`  .run(new VertexOutDegree()`
	`  .setIncludeZeroDegreeVertices(true));`
	可选配置:
	• **setIncludeZeroDegreeVertices**: 默认情况下为了自由度的计算，只有边集 (edge set) 需要被处理；当该参数被设置时，对点集 (vertex set) 会进行一个额外的 join 操作来输出出边数 (out-degree) 为 0 的点
	• **setParallelism**: 指定算子的并行度

degree.annotate.directed.
VertexDegrees
	用自由度（degree），出边（out-degree），和入边（in-degree）标注一个有向图的点.
	`DataSet<Vertex<K, Tuple2<LongValue, LongValue>>> degrees = graph`
	`  .run(new VertexDegrees()`
	`  .setIncludeZeroDegreeVertices(true));`

	可选配置: • **setIncludeZeroDegreeVertices**: 默认情况下为了自由度的计算，只有边集 (edge set) 需要被处理；当该参数被设置时，对点集 (vertex set) 会进行一个额外的 join 操作来输出出边数 (out-degree) 和入边数 (in-degree) 为 0 的点 • **setParallelism**: 指定算子的并行度

degree.annotate.directed.
EdgeSourceDegrees
	用源点的自由度（degree），出边（out-degree）和入边（in-degree）标注一个有向图的边.
	
	`DataSet<Edge<K, Tuple2<EV, Degrees>>> sourceDegrees = graph`
	`  .run(new EdgeSourceDegrees());`
	
	可选配置:
	
	• **setParallelism**: 指定算子的并行度/p>

degree.annotate.directed.
EdgeTargetDegrees
	用目标点的自由度（degree），出边（out-degree）和入边（in-degree）标注一个有向图的边.
	
	`DataSet<Edge<K, Tuple2<EV, Degrees>>> targetDegrees = graph`
	`  .run(new EdgeTargetDegrees();`
	
	可选配置:
	
	• **setParallelism**: 指定算子的并行度

degree.annotate.directed.
EdgeDegreesPair
	用源点目标点的自由度（degree），出边（out-degree）和入边（in-degree）标注一个有向图的边.
	
	`DataSet<Edge<K, Tuple2<EV, Degrees>>> degrees = graph`
	`  .run(new EdgeDegreesPair());`
	
	可选配置:
	
	• **setParallelism**: 指定算子的并行度

degree.annotate.undirected.
VertexDegree
	用自由度（degree）标注一个无向图的点.
	
	`DataSet<Vertex<K, LongValue>> degree = graph`
	`  .run(new VertexDegree()`
	`  .setIncludeZeroDegreeVertices(true)`
	`  .setReduceOnTargetId(true));`
	
	可选配置:
	
	• **setIncludeZeroDegreeVertices**: 默认情况下为了自由度的计算，只有边集 (edge set) 需要被处理；当该参数被设置时，对点集 (vertex set) 会进行一个额外的 join 操作来输出自由度 (degree) 为 0 的点

	• **setParallelism**: 指定算子的并行度 • **setReduceOnTargetId**: 自由度能够用边的源点和终点计算. 默认情况下用源点计算. 如果用目标点对输入边列 (edge list) 排序，对终点的归约可能优化该算法.

degree.annotate.undirected.
EdgeSourceDegree
	用源点的自由度（degree）标注一个无向图的边.
	`DataSet<Edge<K, Tuple2<EV, LongValue>>> sourceDegree = graph`
	`  .run(new EdgeSourceDegree()`
	`  .setReduceOnTargetId(true));`
	可选配置:
	• **setParallelism**: 指定算子的并行度
	• **setReduceOnTargetId**: 自由度能够用边的源点和终点计算. 默认情况下用源点计算. 如果用目标点对输入边列 (edge list) 排序，对终点的归约可能优化该算法.

degree.annotate.undirected.
EdgeTargetDegree
	用目标点的自由度（degree）标注一个无向图的边.
	`DataSet<Edge<K, Tuple2<EV, LongValue>>> targetDegree = graph`
	`  .run(new EdgeTargetDegree()`
	`  .setReduceOnSourceId(true));`
	可选配置:
	• **setParallelism**: 指定算子的并行度
	• **setReduceOnSourceId**: 自由度能够用边的源点和终点计算. 默认情况下用源点计算. 如果用目标点对输入边列 (edge list) 排序，对终点的归约可能优化该算法.

degree.annotate.undirected.
EdgeDegreePair
	用源点和目标点的自由度（degree）标注一个无向图的边.
	`DataSet<Edge<K, Tuple3<EV, LongValue, LongValue>>> pairDegree =`
	`graph`
	`  .run(new EdgeDegreePair()`
	`  .setReduceOnTargetId(true));`
	可选配置:
	• **setParallelism**: 指定算子的并行度
	• **setReduceOnTargetId**: 自由度能够用边的源点和终点计算. 默认情况下用源点计算. 如果用目标点对输入边列 (edge list) 排序，对终点的归约可能优化该算法.

degree.filter.undirected. **MaximumDegree**	用最大自由度过滤一个无向图. `Graph<K, VV, EV> filteredGraph = graph`

	`.run(new MaximumDegree(5000)`
	`.setBroadcastHighDegreeVertices(true)`
	`.setReduceOnTargetId(true));`
	
	可选配置:
	
	• **setBroadcastHighDegreeVertices**: 当移除少量高自由度 (high-degree) 的点时，用一个广播哈希 (broadcast-hash) 合并 (join) 高自由度的点来减少数据洗牌 (shuffle).
	• **setParallelism**: 指定算子的并行度
	• **setReduceOnTargetId**: 自由度能够用边的源点和终点计算. 默认情况下用源点计算. 如果用目标点对输入边列 (edge list) 排序，对终点的归约可能优化该算法.

simple.directed. **Simplify**	移除一个有向图的自环 (self-loops) 和相同的边. `graph.run(new Simplify());` 可选配置: • **setParallelism**: 指定算子的并行度
simple.undirected. **Simplify**	从一个无向图中添加对称边并移除自环 (self-loops). `graph.run(new Simplify());` 可选配置: • **setParallelism**: 指定算子的并行度
translate. **TranslateGraphIds**	用给定的 `TranslateFunction` 转换 (translate) 点和边的 ID. `graph.run(new TranslateGraphIds(new LongValueToStringValue()));` 必要配置: • **translator**: 实现类型或值转换 可选配置: • **setParallelism**: 指定算子的并行度
translate. **TranslateVertexValues**	用给定的 `TranslateFunction` 转换 (translate) 点的值. `graph.run(new TranslateVertexValues(new LongValueAddOffset(vertexCount)));` 必要配置:

	• **translator**: 实现类型或值转换 可选配置: • **setParallelism**: 指定算子的并行度
translate. **TranslateEdgeValues**	用给定的 `TranslateFunction` 转换 (translate) 边的值. `graph.run(new TranslateEdgeValues(new Nullify()));` 必要配置: • **translator**: 实现类型或值转换 可选配置: • **setParallelism**: 指定算子的并行度