SILICON VALLEY
DATA SCIENCE

SPARK SUMMIT 2017

WRITE GRAPH ALGORITHMS LIKE A BOSS

Dr. Andrew Ray

ANDREW RAY

@collegeisfun

PhD Mathematics @ Nebraska

- Graph Theory

Spark contributor

- SQL
- GraphX

Principal Data Engineer @ SVDS

Previously @ Walmart

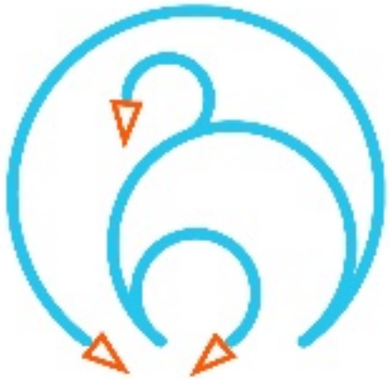
Silicon Valley Data Science is a boutique consulting firm focused on transforming your business through data science and engineering.

OUR SERVICES

DATA STRATEGY

ARCHITECTURE

AGILE ENGINEERING



AGILE DATA SCIENCE

To view SVDS speakers and scheduling, or to receive a copy of our slides, go to:

www.svds.com/SparkSummit17

OUTLINE

- Graph-parallel
- Examples
- Pregel
- PowerGraph
- GraphX

There will be code

Problem Space

GRAPH-PARALLEL

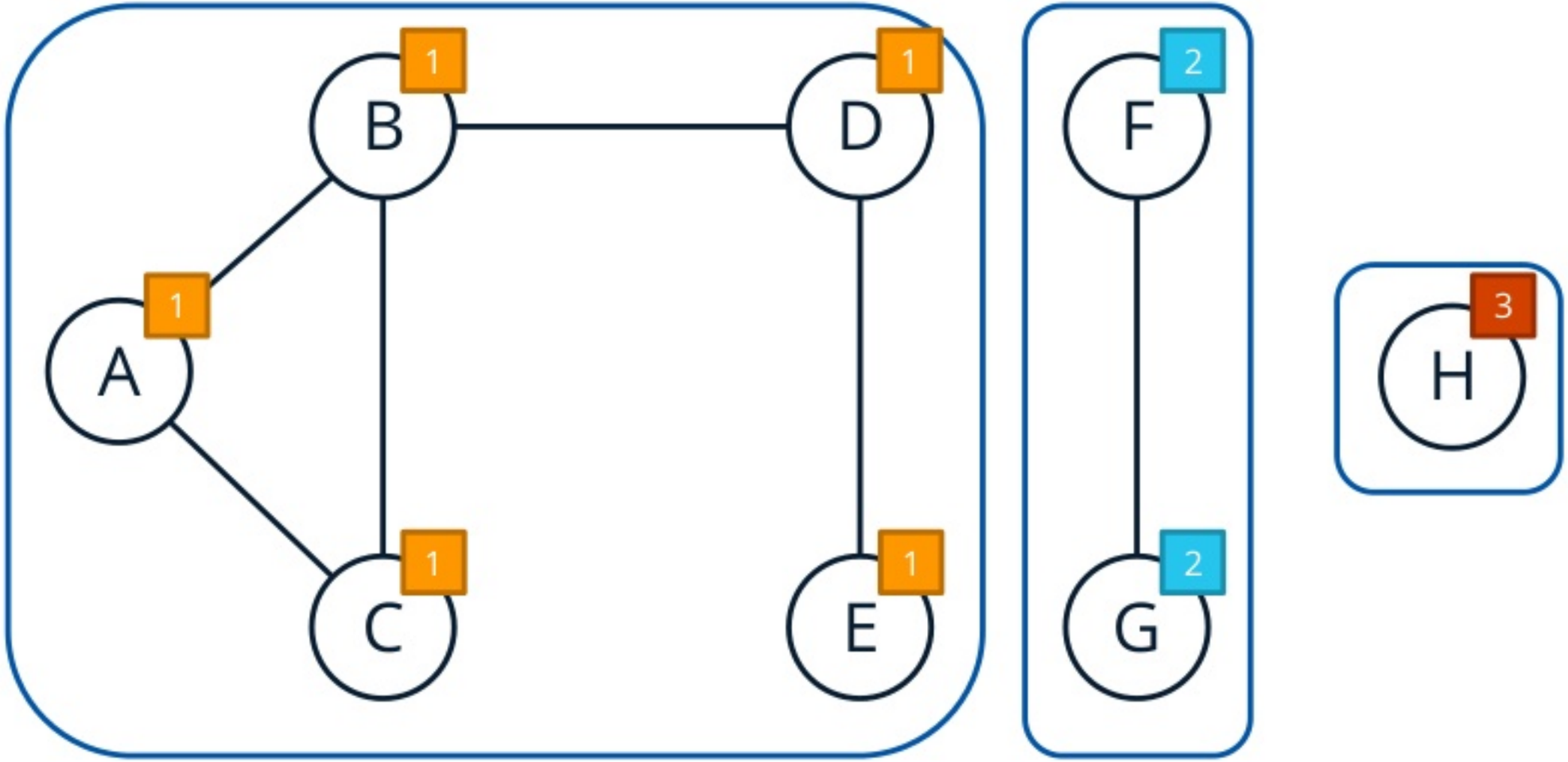
GRAPH-PARALLEL

- Data-parallel for graphs
- Use the entire graph
- Can be worked on from a local context in parallel
- Usually compute a value for every vertex/edge
- Needed for large graphs on distributed systems
- **Pregel, PowerGraph, and GraphX provide graph-parallel abstractions**

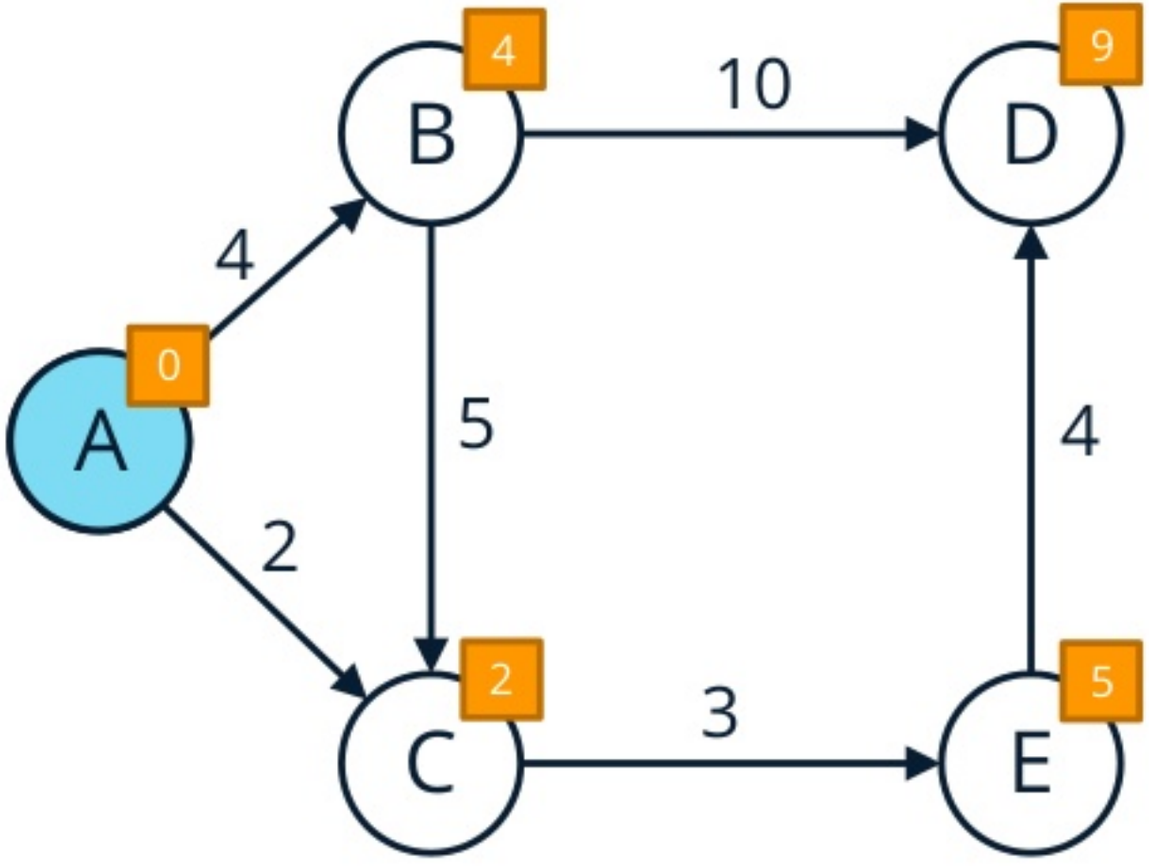
EXAMPLES

- Connected Components
- Single Source Shortest Path
- PageRank

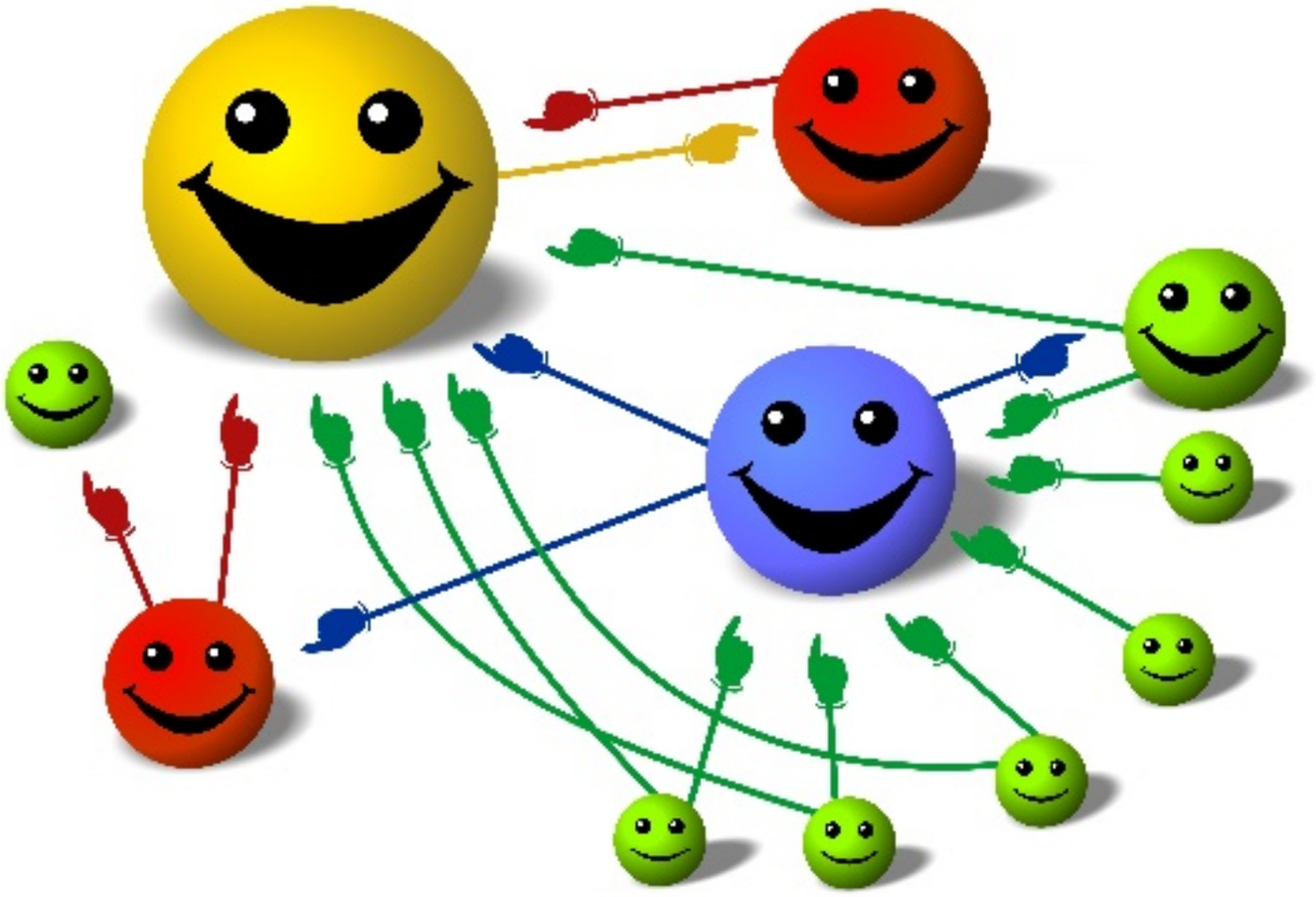
CONNECTED COMPONENTS



SINGLE SOURCE SHORTEST PATH



PAGERANK



Think like a vertex

PREGEL

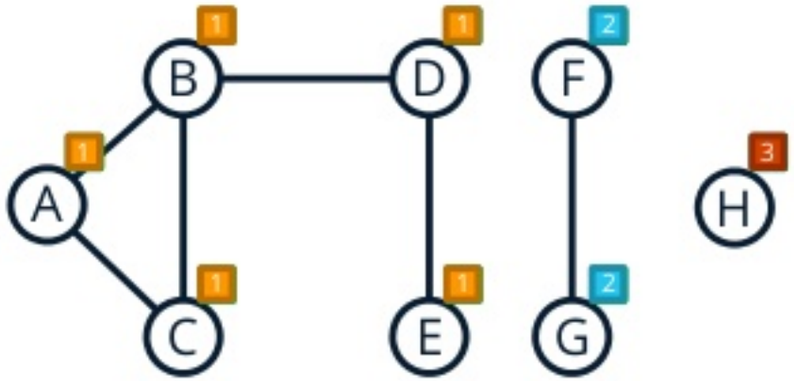
PREGEL
GOOGLE 2010

- Message passing with "supersteps"
- Vertex program
  - Processes messages
  - Updates vertex state
  - Sends messages
  - Can vote to halt
- Optional message combiner
- Global aggregators

CONNECTED COMPONENTS

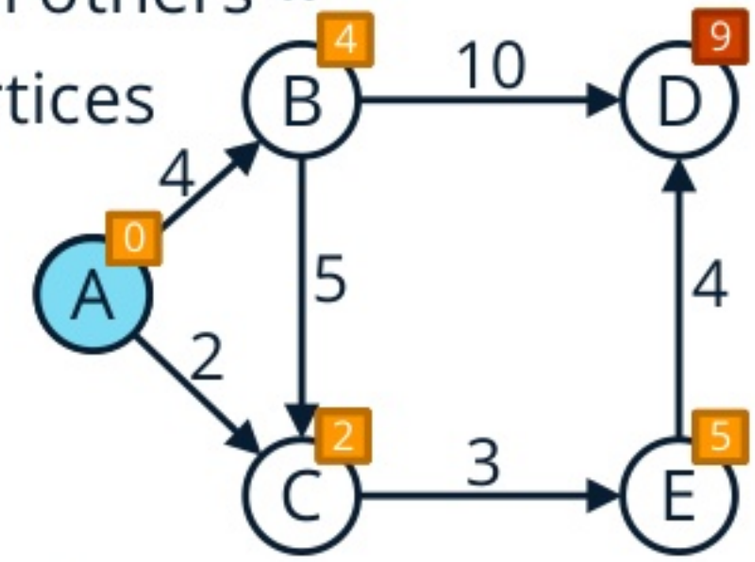
- Initial vertex state: Unique integer vertex id

```
compute(messages):
    min_label = min(messages)
    if min_label < state:
        state = min_label
    else:
        vote_to_halt()
    send_message_to_all_neighbors(state)
```



SINGLE SOURCE SHORTEST PATH

- Initial vertex state: source is 0, all others ∞
- Edge data: distance between vertices




```
compute(messages):
    min_dist = min(messages)
    if min_dist < state:
        state = min_dist
        for neighbor in neighborhood:
            send_message(neighbor, state + edge_weight)
    else:
        vote_to_halt()
```

PAGERANK

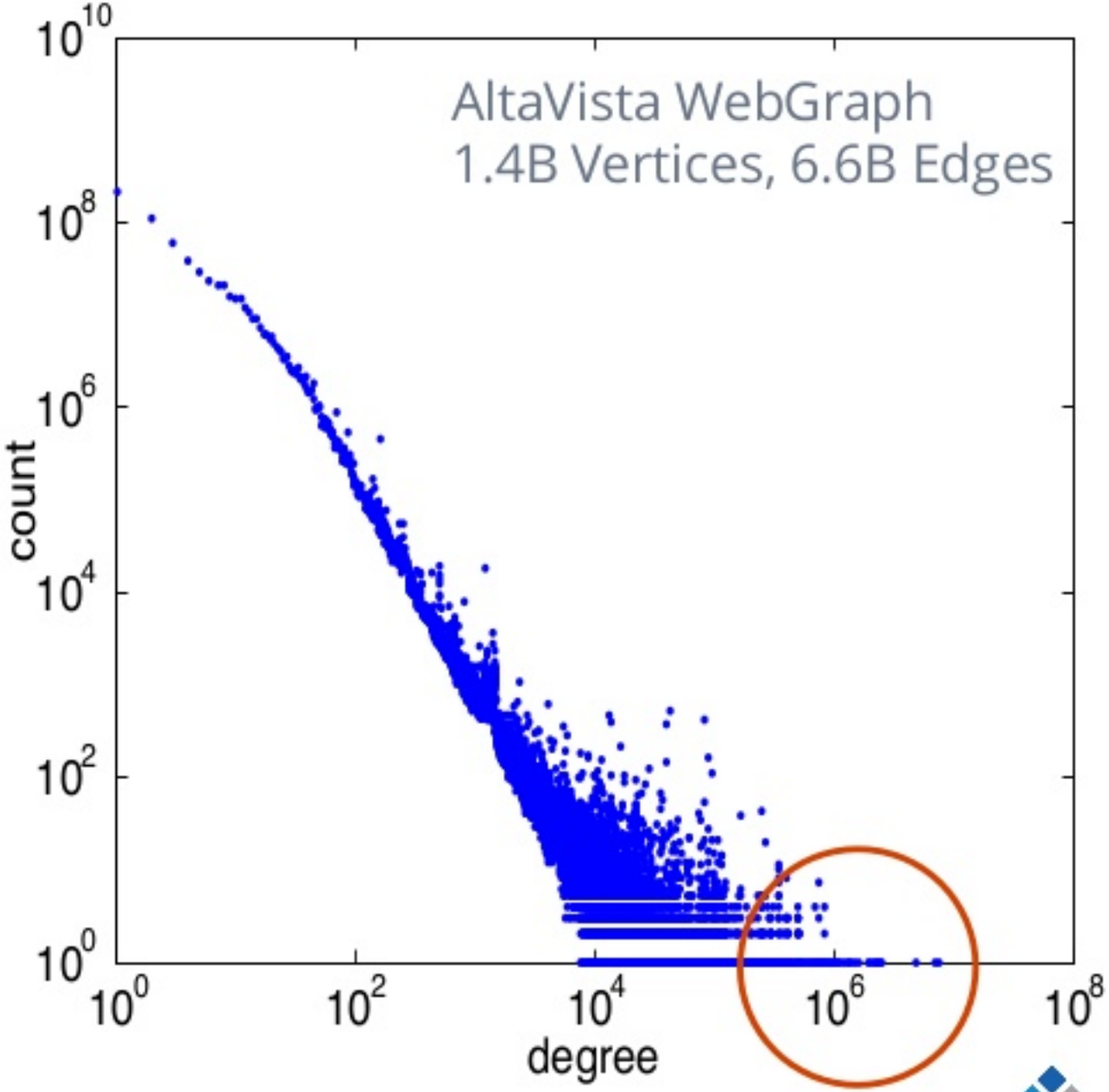
- Initial vertex state: 1

```
compute(messages):
    if superstep > 0:
        sum_messages = sum(messages)
        state = 0.15 + 0.85 * sum_messages
    if superstep < 30:
        send_message_to_all_neighbors(state/out_deg)
    else:
        vote_to_halt()
```

POWERGRAPH

Think like an edge

REAL WORLD DEGREE DIST.



POWERGRAPH CMU 2012

- Gather(D_u, $D_{(u,v)}$, D_v)
  - Function applied to each in/out/both edge incident to *u*
  - Data from both vertices
- sum – combiner
  - Combines gathered values for *u*
- Apply(D_u, *sum*)
  - Update vertex data D_u
- Scatter(D_u, $D_{(u,v)}$, D_v)
  - Update neighboring edge data
  - Activate neighbors
  - Optionally send a delta to *v*

CONNECTED COMPONENTS

- Initial vertex data: Unique integer vertex id

```
gather(D_u, D_(u,v), D_v)[all_nbrs] = D_v
sum(a, b) = min(a, b)
apply(D_u, sum) = min(D_u, sum)
scatter(D_u, D_(u,v), D_v)[all_nbrs] =
    if D_v > D_u:
        activate(v)
```

SINGLE SOURCE SHORTEST PATH

- Initial vertex data: source is 0, all others ∞
- Edge data: distance between vertices

```
gather(D_u, D_(u,v), D_v)[in_nbrs] = D_v + D_(u,v)
sum(a, b) = min(a, b)
apply(D_u, sum) = min(D_u, sum)
scatter(D_u, D_(u,v), D_v)[out_nbrs] =
    if changed(D_u):
        activate(v)
```

PAGERANK

- Initial vertex data: 1

```
gather(D_u, D_(u,v), D_v)[in_nbrs] = D_v / out_deg(v)
sum(a, b) = a + b
apply(D_u, sum) = 0.15 + 0.85 * sum
scatter(D_u, D_(u,v), D_v)[out_nbrs] =
    if |D_u.delta|> epsilon:
        activate(v)
```

GRAPHX

Apache Spark

GRAPHX AMPLAB 2013

- General Graph API built on Spark RDD's
  - Vertex RDD
  - Edge RDD
- Implementation of "Pregel"
  - Messages from edge triplet
  - Requires combiner
  - Implicit activation
- Lower level "Aggregate Messages" building block
  - Used to implement Pregel
- Normal RDD methods

Warning: Scala code ahead

“Pregel” in GraphX

```
def pregel[A: ClassTag](
    initialMsg: A,
    maxIterations: Int = Int.MaxValue,
    activeDirection: EdgeDirection = EdgeDirection.Either)(
    vprog: (VertexId, VD, A) => VD,
    sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
    mergeMsg: (A, A) => A)
  : Graph[VD, ED]
```

CONNECTED COMPONENTS

```
// Simplified version of org.apache.spark.graphx.lib.ConnectedComponents
def connectedComponents[VD: ClassTag, ED: ClassTag](graph: Graph[VD, ED]):
  Graph[VertexId, ED] = {
  graph
    .mapVertices { case (id, attr) => id }
    .pregel(initialMsg = Long.MaxValue)(
      vprog = (id, attr, msg) => math.min(attr, msg),
      sendMsg = edge => {
        if (edge.srcAttr < edge.dstAttr)
          Iterator((edge.dstId, edge.srcAttr))
        else if (edge.srcAttr > edge.dstAttr)
          Iterator((edge.srcId, edge.dstAttr))
        else
          Iterator.empty
      },
      mergeMsg = (a, b) => math.min(a, b)
    )
}
```

SINGLE SOURCE SHORTEST PATH

```
// Based on example in https://spark.apache.org/docs/latest/graphx-programming-guide.html
// see also org.apache.spark.graphx.lib.ShortestPaths
def sssp[VD: ClassTag](graph: Graph[VD, Double], sourceId: VertexId):
  Graph[Double, Double] = {
  graph
    .mapVertices { (id, attr) =>
      if (id == sourceId) 0.0
      else Double.PositiveInfinity
    }.pregel(initialMsg = Double.PositiveInfinity)(
      vprog = (id, dist, msg) => math.min(dist, msg),
      sendMsg = edge => {
        if (edge.srcAttr + edge.attr < edge.dstAttr)
          Iterator((edge.dstId, edge.srcAttr + edge.attr))
        else
          Iterator.empty
      },
      mergeMsg = (a, b) => math.min(a, b)
    )
}
```

“Aggregate Messages” in GraphX

```
def aggregateMessages[A: ClassTag](
    sendMsg: EdgeContext[VD, ED, A] => Unit,
    mergeMsg: (A, A) => A,
    tripletFields: TripletFields = TripletFields.All)
  : VertexRDD[A]
```

PAGERANK

```
// Simplified version of org.apache.spark.graphx.lib.PageRank
def pageRank[VD: ClassTag, ED: ClassTag](graph: Graph[VD,ED], numIter: Int):
  Graph[Double, Double] = {
  var rankGraph: Graph[Double, Double] = graph
    .outerJoinVertices(graph.outDegrees) { (id, vdata, deg) => deg.getOrElse(0) }
    .mapTriplets(e => 1.0 / e.srcAttr)
    .mapVertices((id, attr) => 1.0)
  for (i <- 1 to numIter) {
    val rankUpdates = rankGraph.aggregateMessages[Double](
      sendMsg = ctx => ctx.sendToDst(ctx.srcAttr * ctx.attr),
      mergeMsg = (a, b) => a + b
    )
    rankGraph = rankGraph.outerJoinVertices(rankUpdates) {
      (id, oldRank, msgSumOpt) => 0.15 + 0.85 * msgSumOpt.getOrElse(0.0)
    }
  }
  rankGraph
}
```

Almost done...

GRAPHX TIPS AND TRICKS

- Specify `activeDirection` for Pregel algorithms.
- Specify `tripletFields` for aggregate messages.
- Specify a `PartitionStrategy` (like `EdgePartition2D`) with the appropriate number of partitions.
- Use immutable data types for vertex/edge data.
- Don't use this example code for anything serious.

REFERENCES

- Pregel: A System for Large-Scale Graph Processing (2010)
- PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs (2012)
- GraphX: A Resilient Distributed Graph System on Spark (2013)

SILICON VALLEY
DATA SCIENCE

info@svds.com
@SVDataScience

THANK YOU

Andrew Ray
andrew@svds.com