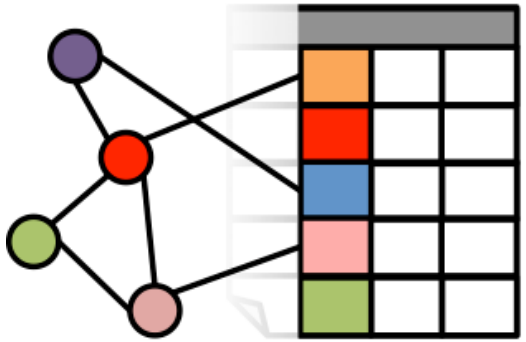
Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

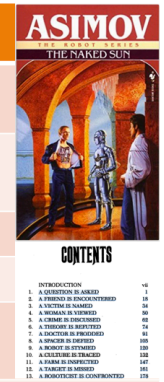
strataconf.com

#StrataHadoop



GraphX

Agenda-Topic	Time	Data	Description
1. Graph Processing Is Introduced	10 Min		Graph Processing vs GraphDB, Introduction to Spark GraphX
2. Basics are Discussed & A Graph Is Built	10 Min	Simple Graph	Edge,Vertex, Graph
3. GraphX API Landscape Is discussed	5 Min		Explain APIs
4. Graph Structures Are Examined	5 Min		Indegree, outdegree et al
5. Community, Affiliation & Strengths Are Explored	5 Min		Connected components, Triangle et al
6. Algorithms Are Developed	10 Min		PartitionStrategy (is what makes GraphParallel work), PageRank, Connected Components; APIs to implement Algorithms viz. aggregateMessages, Pregel Type Signature – aggregateMessages Algorithms with aggregateMessages
7. Case Study (AlphaGo Tweets) Is Conducted	10 Min	AlphaGo Retweet Data	Pipeline, Map attributes to properties, vertices & edges, create graph and run algorithms
8. Questions Are Asked & Answered	10 Min		



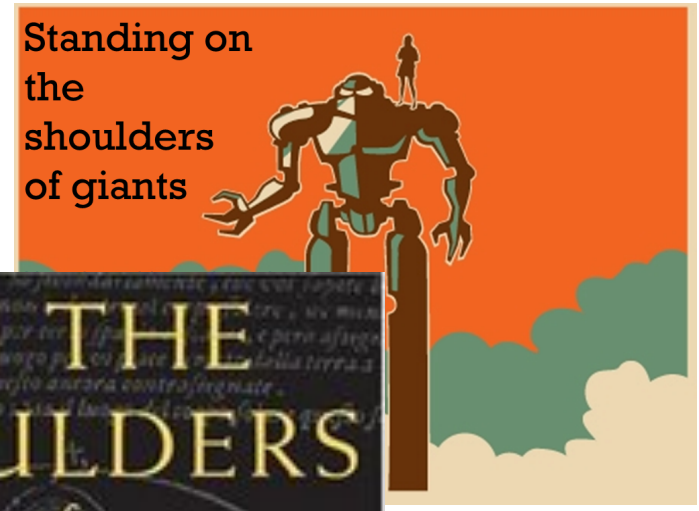
AGENDA TITLES STYLED AFTER ASIMOV'S ROBOT SERIES !!!

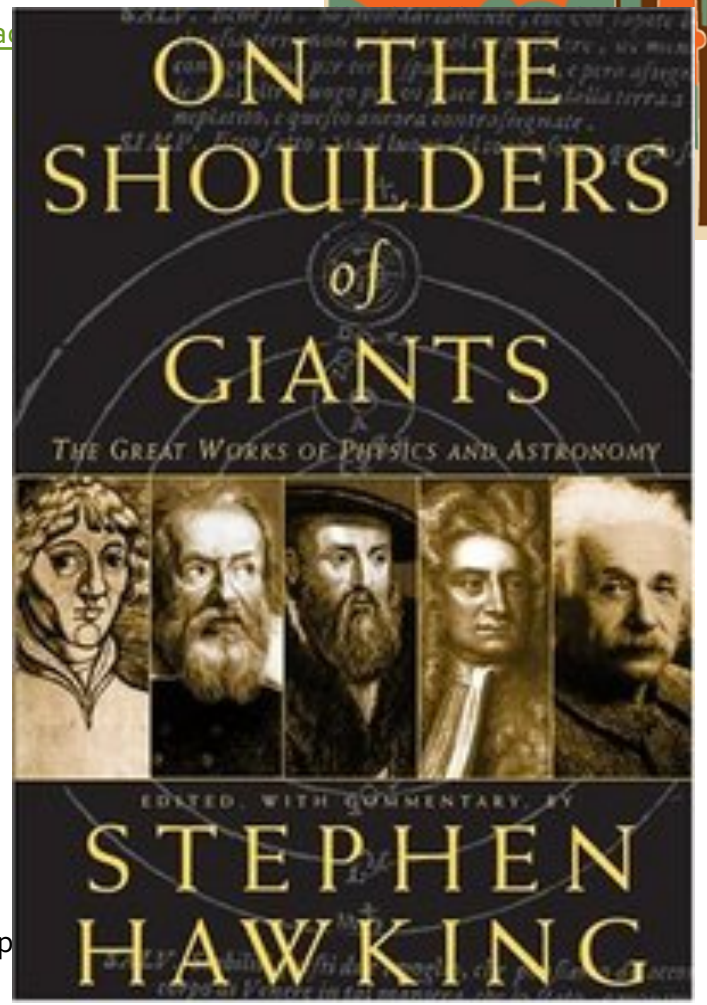
KRISHNA SANKAR
HTTPS://WWW.LINKEDIN.COM/IN/KSANKAR

JUNE 1, 2016

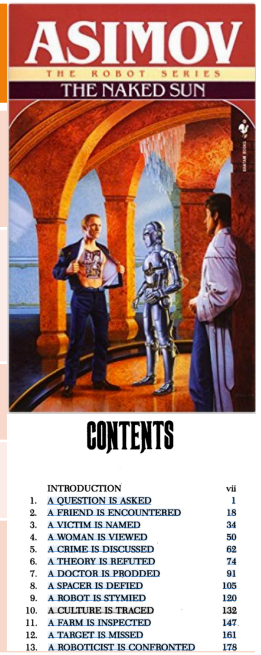
Thanks to the Giants whose work helped to prepare this tutorial

1. Paco Nathan, Scala Days 2015, https://www.youtube.com/watch?v=P_V71n-gtDs
2. Big Data Analytics with Spark-Apress http://www.amazon.com/Big-Data-Analytics-Spark-Practitioners/dp/1484209656
3. Apache Spark Graph Processing-Packt https://www.packtpub.com/big-data-and-business-intelligence/apache-spark-graph-processing
4. Spark GarphX in Action - Manning
5. http://hortonworks.com/blog/introduction-to-data-science-with-apache-spark/
6. http://stanford.edu/~rezab/nips2014workshop/slides/ankur.pdf
7. Mining Massive Datasets book v2 http://infolab.stanford.edu/~ullman/mmds/ch10.pdf
   - http://web.stanford.edu/class/cs246/handouts.html
8. http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm
9. http://kukuruku.co/hub/algorithms/social-network-analysis-spark-graphx
10. Zeppelin Setup
    - http://sparktutorials.net/setup-your-zeppelin-notebook-for-data-science-in-apache-spark
11. Data
    - http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time
    - http://openflights.org/data.html
12. https://adventuresincareerdevelopment.files.wordpress.com/2015/04/standing-on-the-shoulders-of-giants.p





Agenda-Topic	Time	Data	Description
1. Graph Processing Is Introduced	10 Min		Graph Processing vs GraphDB, Introduction to Spark GraphX
2. Basics are Discussed & A Graph Is Built	10 Min	Simple Graph	Edge,Vertex, Graph
3. GraphX API Landscape Is discussed	5 Min		Explain APIs
4. Graph Structures Are Examined	5 Min		Indegree, outdegree et al
5. Community, Affiliation & Strengths Are Explored	5 Min		Connected components, Triangle et al
6. Algorithms Are Developed	10 Min		PartitionStrategy (is what makes GraphParallel work), PageRank, Connected Components; APIs to implement Algorithms viz. aggregateMessages, Pregel Type Signature – aggregateMessages Algorithms with aggregateMessages
7. Case Study (AlphaGo Tweets) Is Conducted	10 Min	AlphaGo Retweet Data	E2E Pipeline w/real data, Map attributes to properties, vertices & edges, create graph and run algorithms
8. Questions Are Asked & Answered	10 Min		



AGENDA TITLES STYLED AFTER ASIMOV'S ROBOT SERIES !!!

Graph Applications

- Many applications from social network to understanding collaboration, diseases, routing logistics and others
- Came across 3 interesting applications, as I was preparing the materials !!

1) *Project effectiveness by social network analysis on the projects' collaboration structures : "EC is interested in the impact generated by those projects … analyzing this latent impact of TEL projects by applying social network analysis on the projects' collaboration structures … TEL projects funded under FP6, FP7 and eContentplus, and identifies central projects and strong, sustained organizational collaboration ties."*

2) *Weak ties in pathogen transmission : "structural motif … Giraffe social networks are characterized by social cliques in which individuals associate more with members of their own social clique than with those outside their clique … Individuals involved in weak, between-clique social interactions are hypothesized to serve as bridges by which an infection may enter a clique and, hence, may experience higher infection risk … individuals who engaged in more between-clique associations, that is, those with multiple weak ties, were more likely to be infected with gastrointestinal helminth parasites … "*

3) *Panama papers : The leak presented them with a wealth of information, millions of documents, but no guide to structure … (ie community detection)*

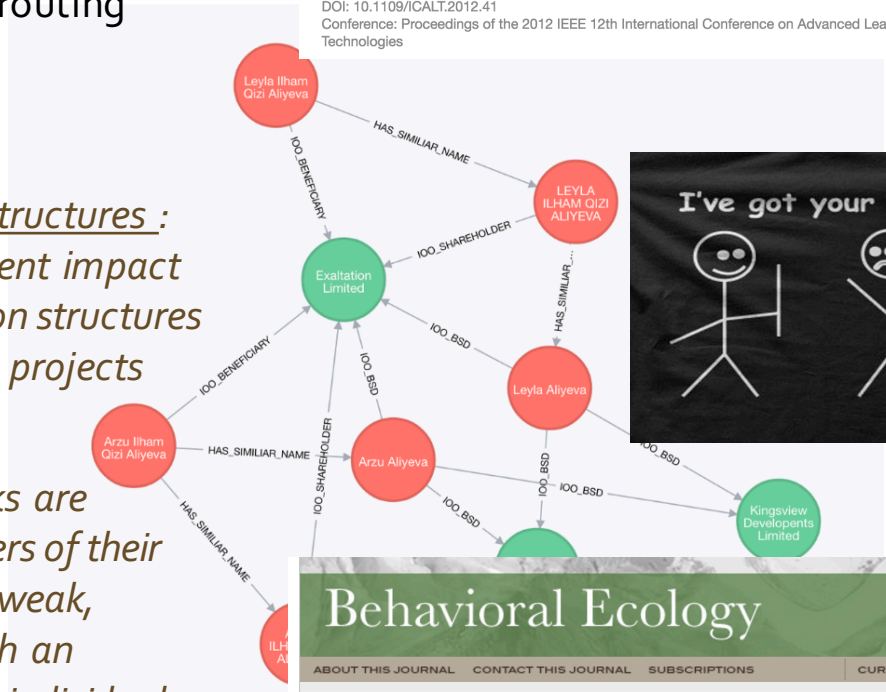
SEE ALL › 1 CITATION SEE ALL › 3 REFERENCES Access full-text

Social Network Analysis of European Project Consortia to Reveal Impact of Technology-Enhanced Learning Projects

CONFERENCE PAPER · JULY 2012 with 168 READS

DOI: 10.1109/ICALT.2012.41

Conference: Proceedings of the 2012 IEEE 12th International Conference on Advanced Learning Technologies



Behavioral Ecology

ABOUT THIS JOURNAL CONTACT THIS JOURNAL SUBSCRIPTIONS CURRENT ISSUE

Oxford Journals › Science & Mathematics › Behavioral Ecology › Advance Access › 10.1093/beheco/arw035

The "strength of weak ties" and helminth parasitism in giraffe social networks

Kimberly L. VanderWaal[a], Vincent Obanda[b], George P. Omondi[c], Brenda McCowan[d,e], Hui Wang[f], Hsieh Fushing[g] and Lynne A. Isbell[h]

[a] Department of Veterinary Population Medicine, University of Minnesota, 1365 Gortner Avenue, St. Paul, MN 55108, USA.

Forbes / Tech

APR 30, 2016 @ 11:36 PM 1,867 VIEWS

Why Panama Papers Journalists Use Graph Databases

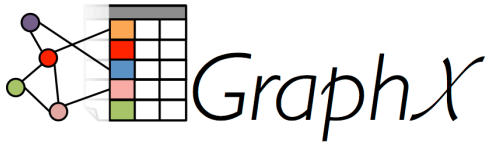
Graph based systems

- Graphs are everywhere – web pages, social networks, people's web browsing behavior, logistics etc.
  - Working w/ graphs has become more important. And harder due to the scale and complexity of algorithms
- Two kinds – processing and querying
- Processing – GraphX, Pregel BSP, GraphLab
- Querying – AllegroGraph, Titan, Neo4j, RDF stores.
  - SPARQL query language
- Graph DBs have queries, can process large dataset
- Graph processing can run complex algorithms
- For Graph Analytics systems both are required
  - Process on graphx, store in neo4j is a good alternative.
    - E.g. Panama papers analytics

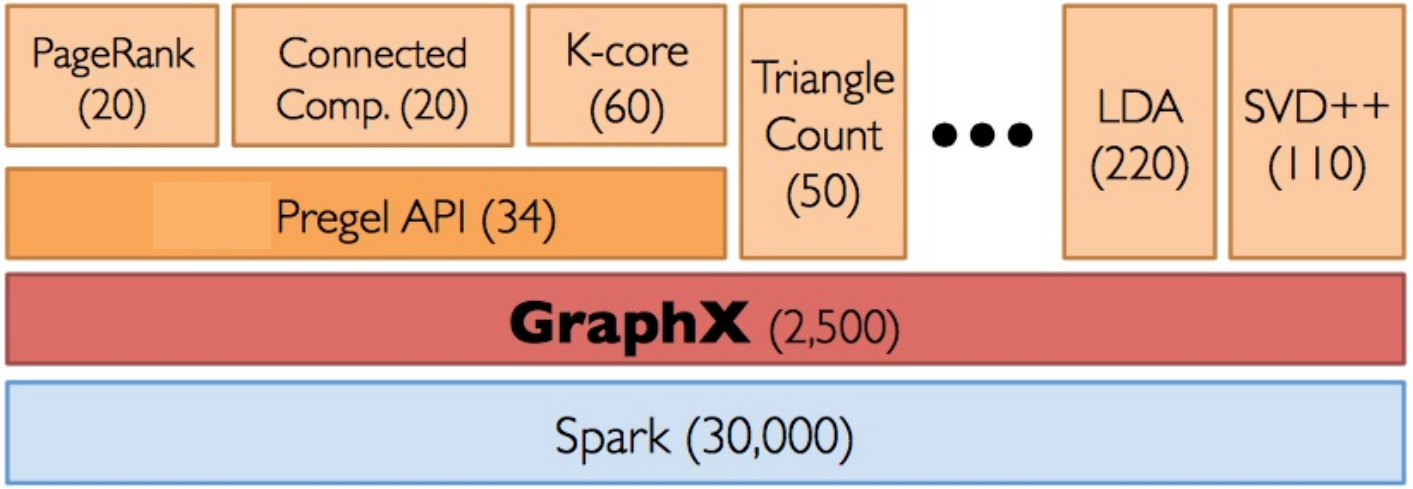
Graph Processing Frameworks

- History – Graph based systems were specialized in nature
  - Graph partitioning is hard ✓ *We will see, later, GrapnX's PartitionStrategy*
  - Primitives of record based systems were different than what graph based systems needed
- Rapid innovation in distributed data processing frameworks – MapReduce etc
  - Disk based, with limited partitioning abilities
  - Still an impedance mismatch
- Graph-parallel over Data Parallel RDD ! – Best of both worlds ?

Enter Spark



- More powerful partitioning mechanism
- In-memory system makes iterative processing easier
- GraphX – Graph processing built on top of Spark
- Graph ***processing*** at scale (distributed system)
- Fast evolving project



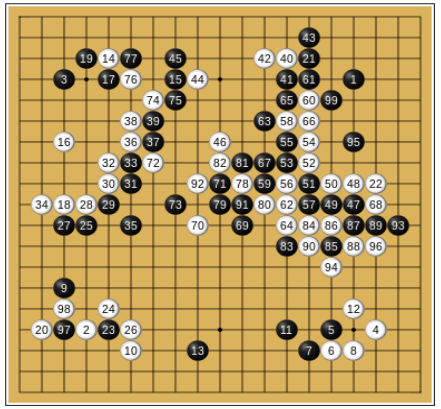
- *Line Count from the Spark Source Code (v1.6.0)*
- *PageRank*
  - *60 lines w/aggregateMessages*
  - *runUntilConvergence – 60 lines w/Pregel*
- *Trianglecount – 50 Lines; SVD++ (150 Lines)*
- *Pragel – lots of comments (> 100 lines/159 -44 lines so 34 is right – example of PageRank w/Pregel API – uses old mapReduceTriplets*
- *LDA in MLLIB/Clustering (~400), - Document, Term = vertices, edges = document -> Term*

GraphX …

- Graph processing at Scale - "Graph Parallel System"
- Has a rich
  - Computational Model
    - Edges, Vertices, Property Graph
    - Bipartite & tripartite graphs (Users-Tags-Web pages)
  - Algorithms (PageRank,…)
- Current focus on computation than query
- APIs include :
  - Attribute Transformers,
  - Structure transformers,
  - Connection Mining & Algorithms
- GraphFrames – interesting development

- Exercises (lots of interesting graph data)
  - Airline data, co-occurrence & co-citation from papers
  - The AlphaGo Community – ReTweet network
  - Wikipedia Page Rank Analysis using GraphX

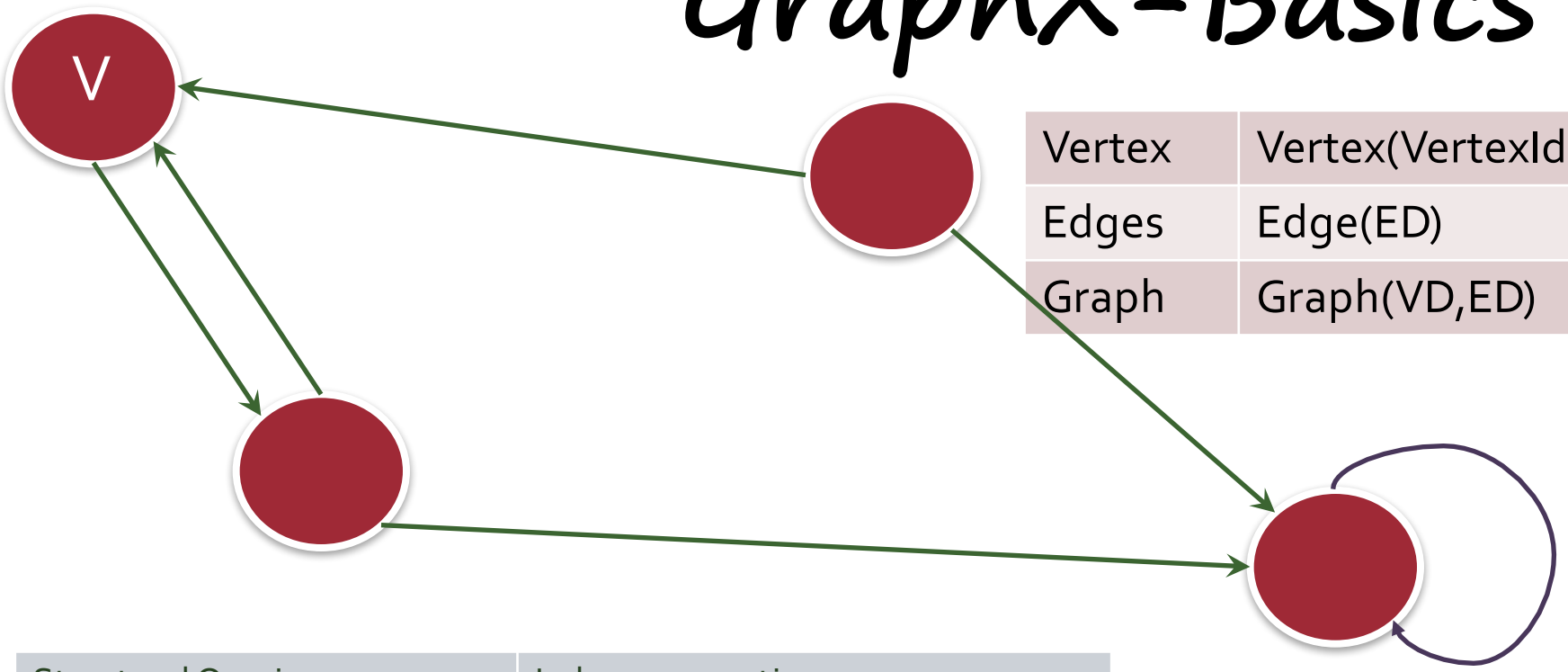
AlphaGo



First 99 moves

Graphx Paper : https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-gonzalez.pdf
Pragel paper : http://kowshik.github.io/JPregel/pregel_paper.pdf
Scala, python support https://issues.apache.org/jira/browse/SPARK-3789
Java API for Graphx https://issues.apache.org/jira/browse/SPARK-3665
LDA https://issues.apache.org/jira/browse/SPARK-1405

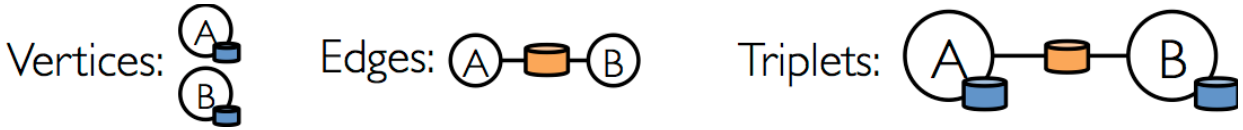
GraphX-Basics



Vertex	Vertex(VertexId,VD)	VD can be any object
Edges	Edge(ED)	ED can be any object
Graph	Graph(VD,ED)	

Structural Queries	Indegrees, vertices,…
Attribute Transformers	mapVertices, …
Structure Transformers, Join	Reverse, subgraph
Connection Mining	connectedComponents, triangles
Algorithms	Aggregate messages, PageRank

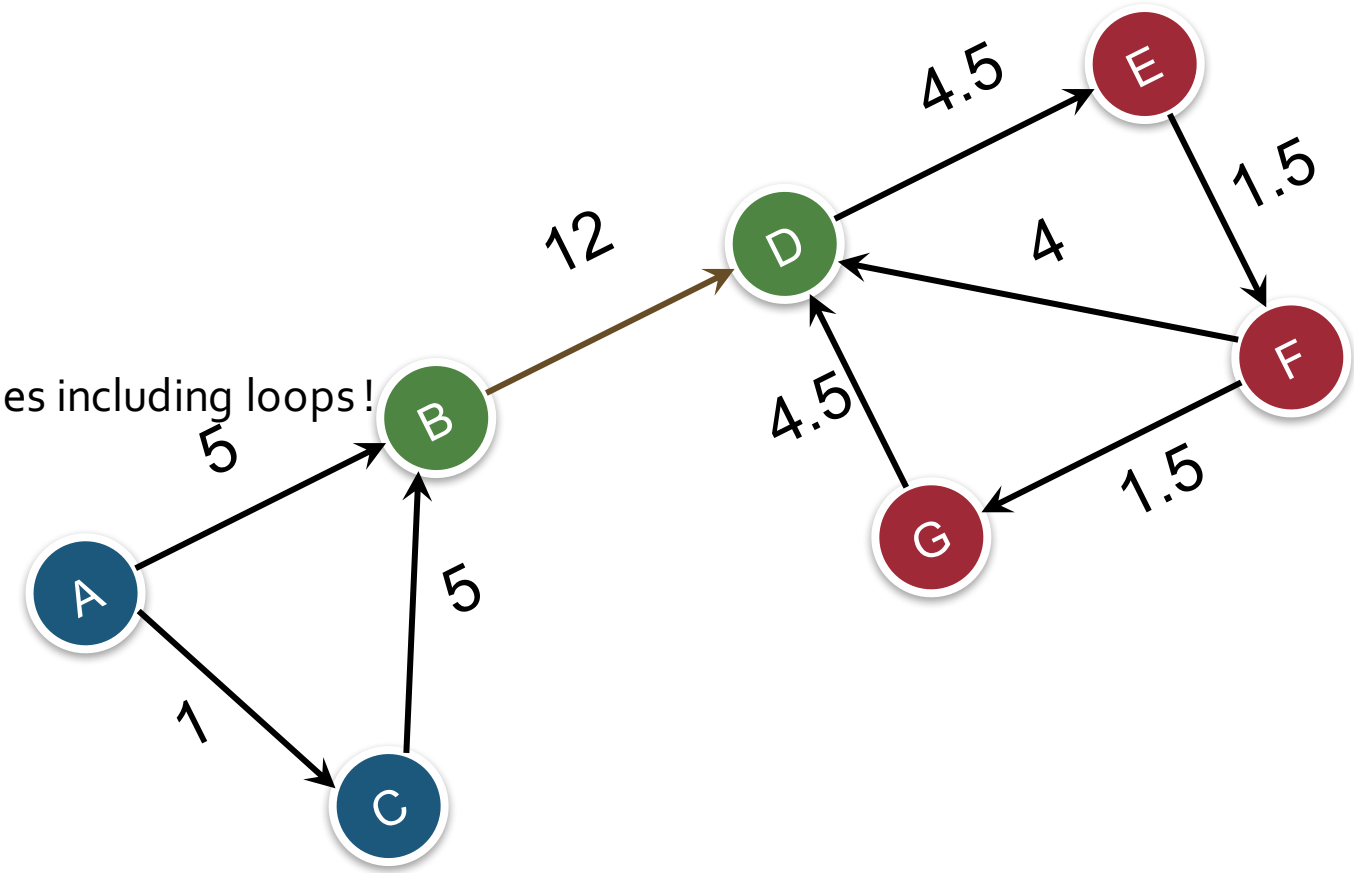
Interesting Objects
- *EdgeTriplet*
- *EdgeContext*



http://ampcamp.berkeley.edu/big-data-mini-course/graph-analytics-with-graphx.html

GraphX-Basics

- Computational Model
  - Directed MultiGraph
    - Directed - so in-degree & out-degree
    - MultiGraph – so multiple parallel edges between nodes including loops !
    - Algorithms beware – can cyclic, loops
  - Property Graph
    - vertexID(64bit long) int
    - Need property, cannot ignore it
    - Vertex,Edge Parameterized over object types
      - Vertex[(VertexId,VD)], Edge[ED]
    - Attach user-defined objects to edges and vertices (ED/VD)

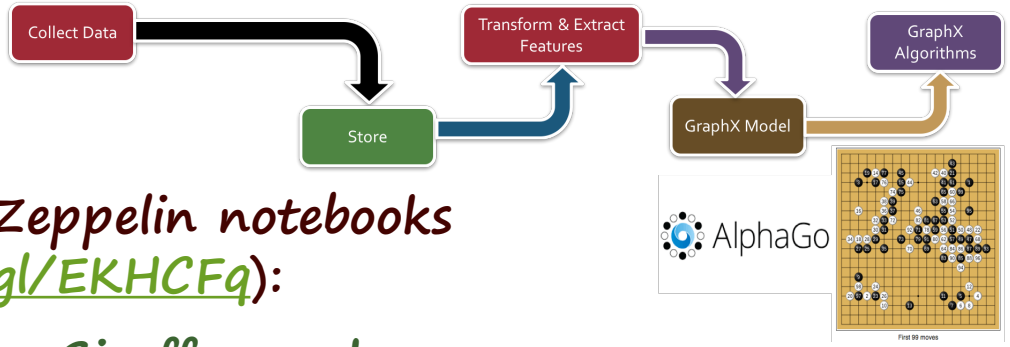


Good exercises https://www.sics.se/~amir/files/download/dic/answers6.pdf

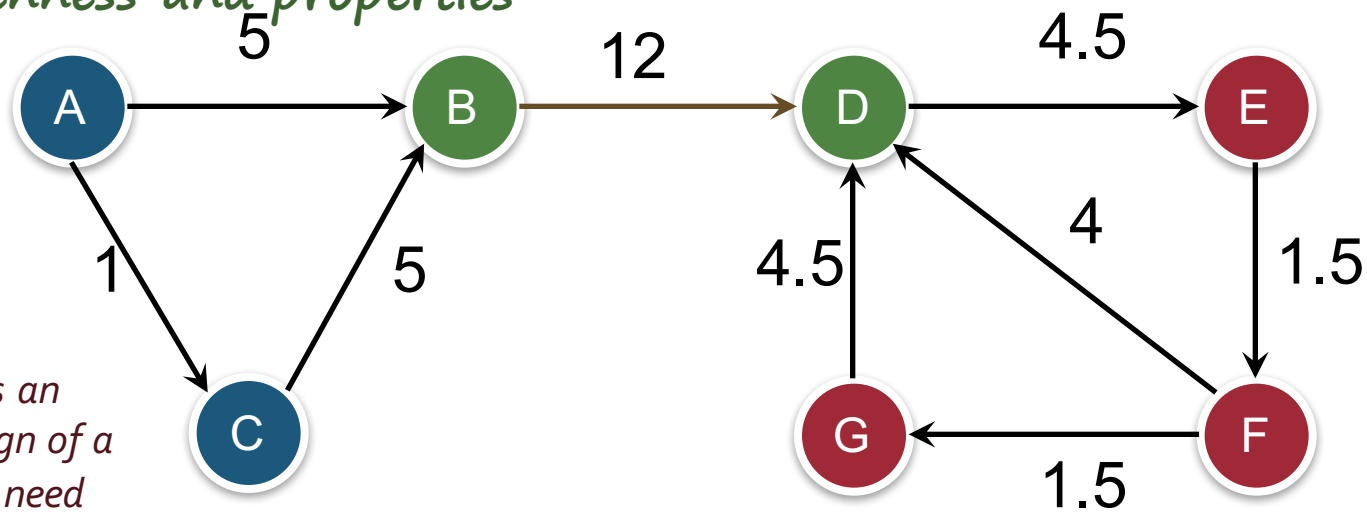
Graphs and Social Networks-Prof. Jeffrey D. Ullman Stanford University
http://web.stanford.edu/class/cs246/handouts.html
G-N Algorithm for inbetweenness

GraphX-Basics

```
val vertexList = List(
(1L, Person("Alice", 18)),
(2L, Person("Bernie", 17)),
(3L, Person("Cruz", 15)),
(4L, Person("Donald", 12)),
(5L, Person("Ed", 15)),
(6L, Person("Fran", 10)),
(7L, Person("Genghis",854)))
```



- **For our hands-on we will run thru 2 Zeppelin notebooks (https://goo.gl/qCwZiq & https://goo.gl/EKHCFq):**
  - **a. First we will work with the following Giraffe graph**
    - … carefully chosen with interesting betweenness and properties
    - … for understanding the APIs
  - **b. Second, we will develop a retweet pipeline**
    - With the AlphaGo twitter topic
    - 2 GB/330K tweets, 200K edges,…



Fun facts about centrality betweenness : It shows how many paths an edge is part of, i.e. relevancy. High centrality betweenness is the sign of a bottleneck, point of single failure – these edges need HA, probably need alternate paths for re routing, are susceptible to parasite infections and good candidates for a cut !

Good exercises https://www.sics.se/~amir/files/download/dic/answers6.pdf

Graphs and Social Networks-Prof. Jeffrey D. Ullman Stanford University
http://web.stanford.edu/class/cs246/handouts.html
G-N Algorithm for inbetweenness

Build A Graph

```
case class Person(name:String,age:Int)

val defaultPerson = Person("NA",0)

val vertexList = List(
  (1L, Person("Alice", 18)),
  (2L, Person("Bernie", 17)),
  (3L, Person("Cruz", 15)),
  (4L, Person("Donald", 12)),
  (5L, Person("Ed", 15)),
  (6L, Person("Fran", 10)),
  (7L, Person("Genghis",854))
)
```

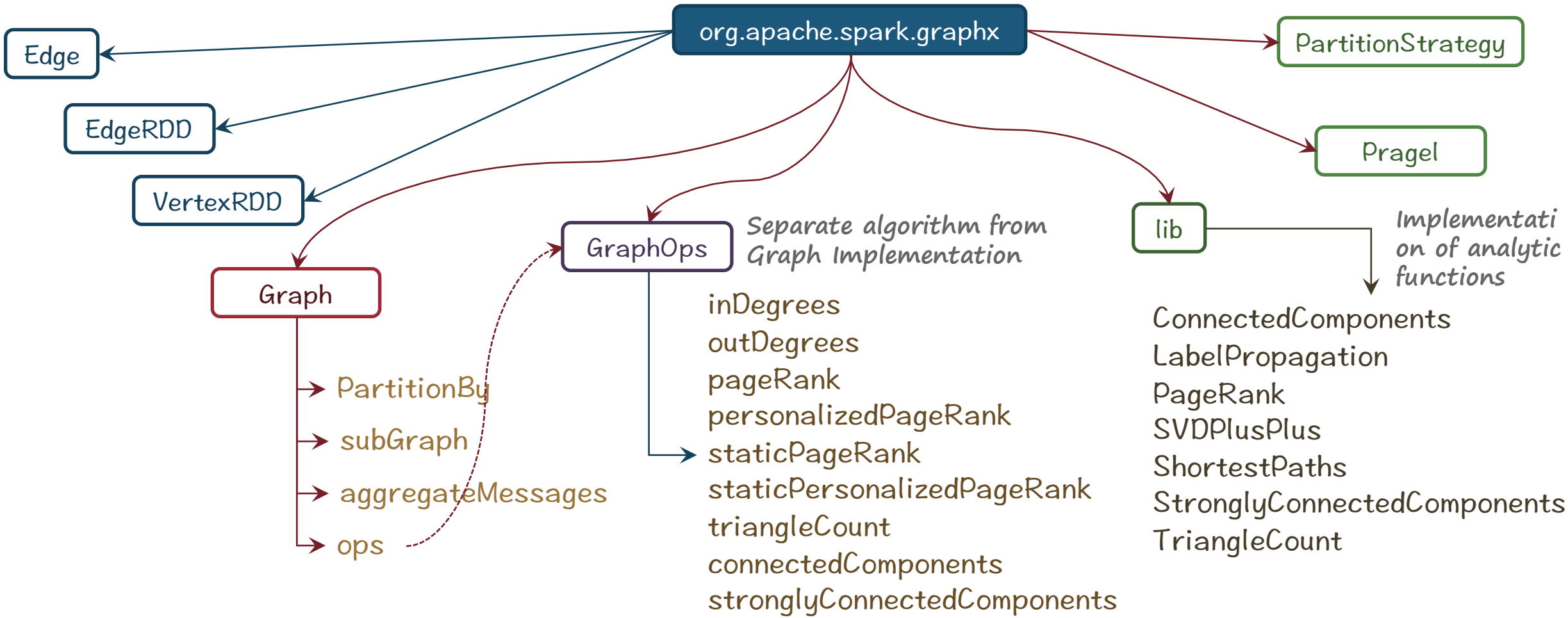
```
val edgeList = List(
    Edge(1L, 2L, 5),
    Edge(1L, 3L, 1),
    Edge(3L, 2L, 5),
    Edge(2L, 4L, 12),
    Edge(4L, 5L, 4),
    Edge(5L, 6L, 2),
    Edge(6L, 7L, 2),
    Edge(7L, 4L, 5),
    Edge(6L, 4L, 4)
    )
```

```
val vertexRDD = sc.parallelize(vertexList)
val edgeRDD = sc.parallelize(edgeList)
val graph = Graph(vertexRDD, edgeRDD,defaultPerson)
//
println("Edges = " + graph.numEdges)
println("Vertices = " + graph.numVertices)
//
val vertices = graph.vertices
vertices.collect.foreach(println)
val edges = graph.edges
edges.collect.foreach(println)
val triplets = graph.triplets
triplets.take(3)
triplets.map(t=>t.toString).collect().foreach(println)
//
```

- There are 4 ways
  - GraphLoader.edgeListFile(…)
  - From RDDs (shown above)
  - fromEdgeTuples <- Id tuples
  - fromEdges <- edgeList

```
# Comment Line
# Source Id <\t> Target Id
1   -5
1    2
2    7
1    8
```

Graph API Landscape



Hint:

Many details are documented in the object not in the class e.g. PartitionStrategy, Graph,...

org.apache.spark.graphx

PartitionStrategy

Related Docs: object PartitionStrategy | package graphx

```
trait PartitionStrategy extends Serializable
```

Represents the way edges are assigned to edge partitions based on their source and destination vertex IDs.

- Linear Supertypes
- Known Subclasses

Ordering: Alphabetic | By Inheritance

Inherited: PartitionStrategy | Serializable | Serializable | AnyRef | Any

Hide All | Show All

Visibility: Public | All

Abstract Value Members

```
abstract def getPartition(src: VertexId, dst: VertexId, numParts: PartitionID): PartitionID
```

Returns the partition number for a given edge.

org.apache.spark.graphx

PartitionStrategy

Related Docs: trait PartitionStrategy | package graphx

```
object PartitionStrategy extends Serializable
```

Collection of built-in PartitionStrategy implementations.

- Linear Supertypes

Ordering: Alphabetic | By Inheritance

Inherited: PartitionStrategy | Serializable | Serializable | AnyRef | Any

Hide All | Show All

Visibility: Public | All

Value Members

```
object CanonicalRandomVertexCut extends PartitionStrategy with Product with Serializable
```

Assigns edges to partitions by hashing the source and destination vertex IDs in a canonical direction, resulting in a random vertex cut that colocates all edges between two vertices, regardless of direction.

```
object EdgePartition1D extends PartitionStrategy with Product with Serializable
```

Assigns edges to partitions using only the source vertex ID, colocating edges with the same source.

```
object EdgePartition2D extends PartitionStrategy with Product with Serializable
```

Assigns edges to partitions using a 2D partitioning of the sparse edge adjacency matrix, guaranteeing a `2 * sqrt(numParts)` bound on vertex replication.

```
object RandomVertexCut extends PartitionStrategy with Product with Serializable
```

Assigns edges to partitions by hashing the source and destination vertex IDs, resulting in a random vertex cut that colocates all same-direction edges between two vertices.

```
def fromString(s: String): PartitionStrategy
```

Returns the PartitionStrategy with the specified name.

Graphx Structure API

```
graph.numEdges
graph.numVertices
```

```
res14: Long = 9
res15: Long = 7
```

Took 1 seconds

```
val vertices = graph.vertices
vertices.collect.foreach(println)
```

```
vertices: org.apache.spark.graphx.Ver
(4,Person(Donald,12))
(1,Person(Alice,18))
(5,Person(Ed,15))
(6,Person(Fran,10))
(2,Person(Bernie,17))
(3,Person(Cruz,15))
(7,Person(Genghis,854))
```

```
val edges = graph.edges
edges.collect.foreach(println)
```

```
edges: org.apache.spark.graphx.E
Edge(1,2,5)
Edge(1,3,1)
Edge(2,4,12)
Edge(3,2,5)
Edge(4,5,4)
Edge(5,6,2)
Edge(6,4,4)
Edge(6,7,2)
Edge(7,4,5)
```

```
val triplets = graph.triplets
triplets.take(3)
triplets.map(t=>t.toString).collect().foreach(println)
```

```
triplets: org.apache.spark.rdd.RDD[org.apache.spark.graphx
res21: Array[org.apache.spark.graphx.EdgeTriplet[Person,In
((2,Person(Bernie,17)),(4,Person(Donald,12)),12))
((1,Person(Alice,18)),(2,Person(Bernie,17)),5)
((1,Person(Alice,18)),(3,Person(Cruz,15)),1)
((2,Person(Bernie,17)),(4,Person(Donald,12)),12)
((3,Person(Cruz,15)),(2,Person(Bernie,17)),5)
((4,Person(Donald,12)),(5,Person(Ed,15)),4)
((5,Person(Ed,15)),(6,Person(Fran,10)),2)
((6,Person(Fran,10)),(4,Person(Donald,12)),4)
((6,Person(Fran,10)),(7,Person(Genghis,854)),2)
((7,Person(Genghis,854)),(4,Person(Donald,12)),5)
```

```
val inDeg = graph.inDegrees // Followers
inDeg.collect()
val outDeg = graph.outDegrees // Follows
outDeg.collect()
val allDeg = graph.degrees
allDeg.collect()
```

```
inDeg: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[26] at RDD at VertexRDD.scala:57
res24: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,3), (5,1), (6,1), (2,2), (3,1), (7,1))
outDeg: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[30] at RDD at VertexRDD.scala:57
res25: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (1,2), (5,1), (6,2), (2,1), (3,1), (7,1))
allDeg: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[34] at RDD at VertexRDD.scala:57
res26: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,4), (1,2), (5,2), (6,3), (2,3), (3,2), (7,2))
```

Took 2 seconds

Community-Affiliation-Strengths

- Applied in many ways
- For example in Fraud & Security Applications
- Triangle detection – for spam servers
- The age of a community is related to the density of triangles
  - New community will have few triangles, then triangles start to form
- Strong affiliation ie Heavy hitter = sqrt(m) degrees!
  - Heavy hitter triangle !
- Connected Communities – structure

What's the most interesting application of Machine Learning that LinkedIn has in place?

Deepak Agarwal, Head of relevance and Machine Learning at LinkedIn
3k Views • Upvoted by William Chen, Data Scientist at Quora

Almost every product at LinkedIn is powered by Machine Learning. Choosing one would be like selecting a favorite kid out of many. I will list some of the most significant ones, but this is my no means an exhaustive list.

1. Feed Relevance
2. People you may know
3. Jobs recommendations
4. Search (people, recruiter, others)
5. Email and notification relevance
6. Advertising
7. Course recommendation

What kind of tools/languages do Machine Learning engineers at LinkedIn use?

Deepak Agarwal, Head of relevance and Machine Learning at LinkedIn
2.7k Views • Upvoted by Nikhil Garg, I lead a team of Quora engineers working on ML/NLP problems

For offline: Pig, scalding, Spark, good old Hadoop, GraphX

Algorithms

- Graph-Parallel Computation
  - aggregateMessages() Function
  - Pregel() (https://issues.apache.org/jira/browse/SPARK-5062)
- pageRank()
  - Influential papers in a citation network
  - Influencer in retweet
- staticPageRank()
  - Static no of iterations, dynamic tolerance – see the parameters (tol vs. numIter)
- personalizedPageRank()
  - Personalized PageRank is a variation on PageRank that gives a rank relative to a specified "source" vertex in the graph – "People You May Know"
- shortestPaths, SVD++

- LabelPropagation (LPA) as described by Raghavan et al in their 2007 paper "Near linear time algorithm to detect community structures in large-scale networks"
  - Computationally inexpensive way to Identify communities
  - Convergence not guaranteed
  - Might end up with trivial solution i.e. single community
- SDVPlusPlus takes an RDD of Edges
- The Global Clustering Coefficient, is better in that it always returns a number between 0 and 1
  - For comparing connectedness between different sized communities

http://graphframes.github.io/user-guide.html

```
def aggregateMessages[A](sendMsg: (EdgeContext[VD, ED, A]) ⇒ Unit, mergeMsg: (A, A) ⇒ A,
    tripletFields: TripletFields = TripletFields.All)(implicit arg0: ClassTag[A]): VertexRDD[A]
```

Aggregates values from the neighboring edges and vertices of each vertex. The user-supplied `sendMsg` function is invoked on each edge of the graph, generating 0 or more messages to be sent to either vertex in the edge. The `mergeMsg` function is then used to combine all messages destined to the same vertex.

A	the type of message to be sent to each vertex
sendMsg	runs on each edge, sending messages to neighboring vertices using the EdgeContext.
mergeMsg	used to combine messages from `sendMsg` destined to the same vertex. This combiner should be commutative and associative.
tripletFields	which fields should be included in the EdgeContext passed to the `sendMsg` function. If not all fields are needed, specifying this can improve performance.

attr: ED
The attribute associated with the edge.

dstAttr: VD
The vertex attribute of the edge's destination vertex.

dstId: VertexId
The vertex id of the edge's destination vertex.

sendToDst(msg: A): Unit
Sends a message to the destination vertex.

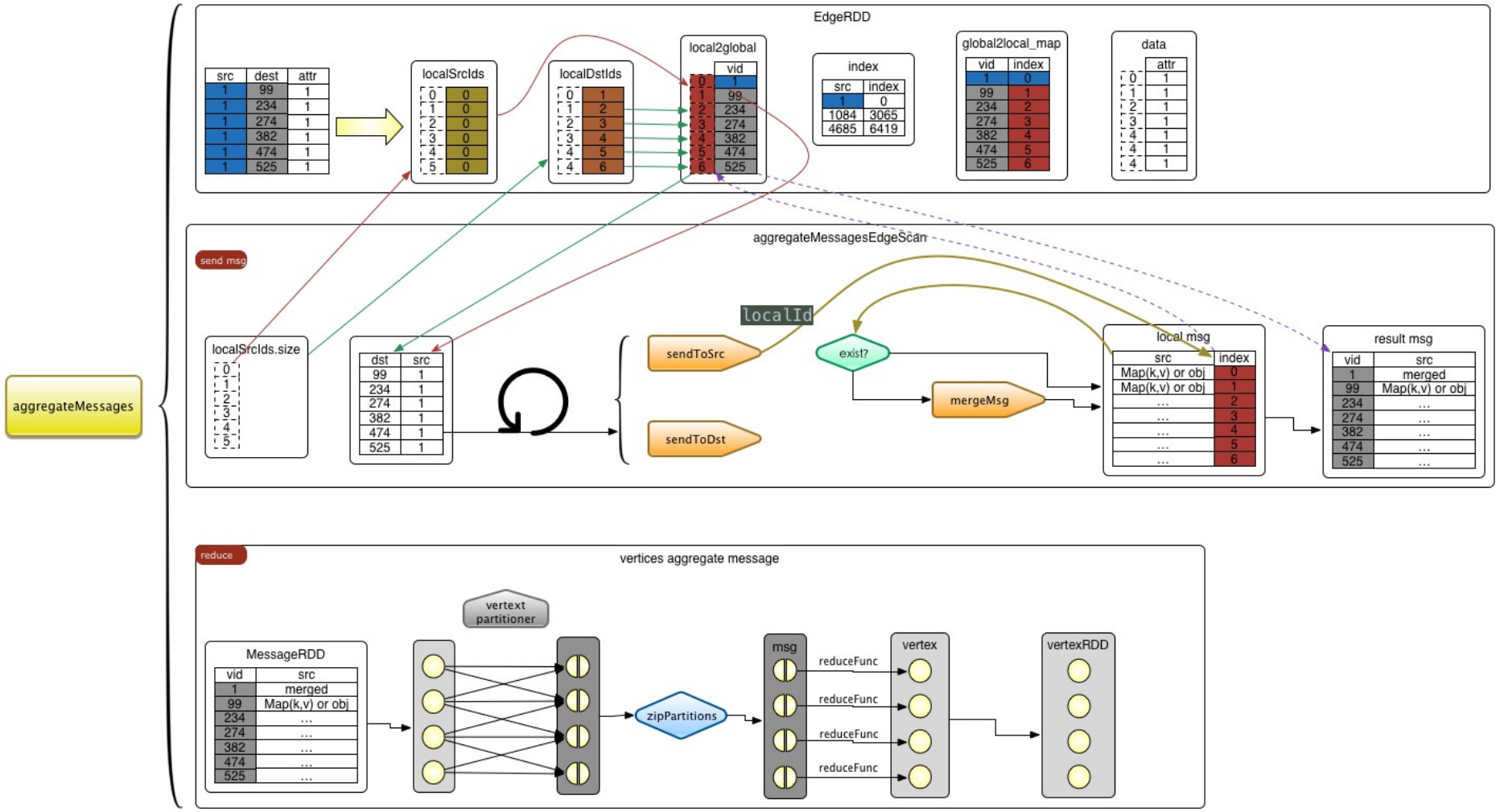
sendToSrc(msg: A): Unit
Sends a message to the source vertex.

srcAttr: VD
The vertex attribute of the edge's source vertex.

srcId: VertexId
The vertex id of the edge's source vertex.

- *Versatile Function useful for implementing PageRank et al*
  - *Can be difficult at first, but easier to comprehend if treated as a combined map-reduce function (it was called MapReduceTriplets !! With a slightly different signature)*
    - aggregateMessage[Msg](
      - map(edgeContext=> mapFun, <- this can be up or down ie sendToDst or Src!
      - redcuce(Msg,Msg) => reduceFun)

If you really want to know what is underneath the aggregateMessages()...



```
val oldestFollower = graph.aggregateMessages[Int](
    edgeContext => edgeContext.sendToDst(edgeContext.srcAttr.age),//sendMsg
    (x,y) => math.max(x,y) //mergeMsg
    )
oldestFollower.collect()
```

```
oldestFollower: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2056] at RDD at VertexRDD.scala:57
res79: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,854), (5,12), (6,15), (2,18), (3,18), (7,10))
```

sendToDst vs sendToSrc

```
val oldestFollowee = graph.aggregateMessages[Int](
    edgeContext => edgeContext.sendToSrc(edgeContext.srcAttr.age),//sendMsg
    (x,y) => math.max(x,y) //mergeMsg
    )
oldestFollowee.collect()
```

```
oldestFollowee: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2092] at RDD at VertexRDD.scala:57
res99: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,12), (1,18), (5,15), (6,10), (2,17), (3,15), (7,854))
```

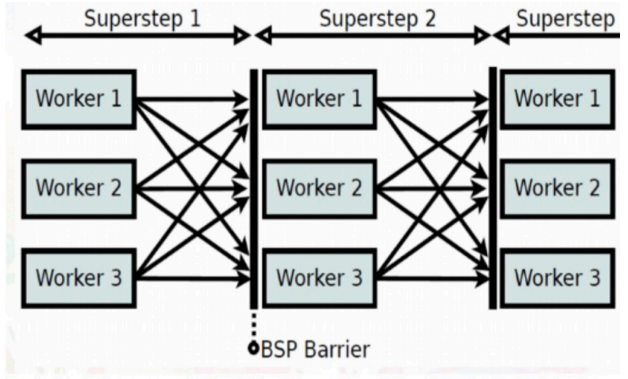
```
val iDegree = graph.aggregateMessages[Int](
    edgeContext => edgeContext.sendToDst(1),//sendMsg
    (x,y) => x+y //mergeMsg
    )
iDegree.collect()
graph.inDegrees.collect()
```

```
iDegree: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2096] at RDD at VertexRDD.scala:57
res101: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,3), (5,1), (6,1), (2,2), (3,1), (7,1))
res102: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,3), (5,1), (6,1), (2,2), (3,1), (7,1))
```

```
val oDegree = graph.aggregateMessages[Int](
    edgeContext => edgeContext.sendToSrc(1),//sendMsg
    (x,y) => x+y //mergeMsg
    )
oDegree.collect()
graph.outDegrees.collect()
```

```
oDegree: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2100] at RDD at VertexRDD.scala:57
res104: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (1,2), (5,1), (6,2), (2,1), (3,1), (7,1))
res105: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (1,2), (5,1), (6,2), (2,1), (3,1), (7,1))
```

Pregel BSP API



Source: https://cs.uwaterloo.ca/~kdaudjee/courses/cs848/slides/jenny.pdf

- Bulk Synchronous Parallel Message Passing API
  - Developed by Leslie Valiant[1]
  - Synchronized distributed super steps
    - Super Step – local, in-memory computations
- Computations on the vertex – "Think like a Vertex"
- graphx.lib.LabelPropagation Algorithm uses Pregel internally
- graphx.lib.PageRank uses Pragel (for the runUntilConvergenceWithOptions version, while it uses the aggregateMessages for the runWithOptions (staticPageRank) version)
- Pregel is implemented succinctly with the old mapReduceTriplets API

[1]http://delivery.acm.org/10.1145/80000/79181/p103-valiant.pdfhttp://www.slideshare.net/riyadparvez/pregel-35504069

Graphx Partitioning & Processing

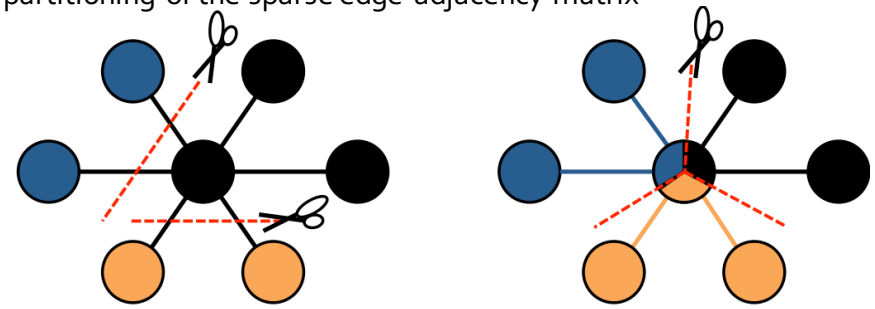
- Unlike a relational table, graph processing is contextual w.r.t a neighborhood
  - Maintain locality, equal size partitioning
  - Edge-cut : The communication & storage overhead of an edge-cut is directly proportional to the number of edges that are cut
  - Vertex-cut : The communication and storage overhead of a vertex-cut is directly proportional to the sum of the number of machines spanned by each vertex
- Vertex-cut strategy by default (balance hot-spot issue due to power law/Zipf's Law) w/ min replication of edges
- Batch processing/not streaming

- Org.apache.spark.graphx.Graph.
  - partitionBy(partitionStrategy: PartitionStrategy, numPartitions: Int)
  - partitionBy(partitionStrategy: PartitionStrategy)
  - 4 PartitionStrategies

1) RandomVertexCut(usually the best) : random vertex cut that colocates all same-direction edges between two vertices (hashing the source and destination vertex IDs)
2) CanonicalRandomVertexCut - a random vertex cut that colocates all edges between two vertices, regardless of direction (hashing the source and destination vertex IDs in a canonical direction) … *remember GraphX is multi graph*
3) EdgePartition1D - Assigns edges to partitions using only the source vertex ID, colocating edges with the same source
4) EdgePartition2D : 2D partitioning of the sparse edge adjacency matrix

groupEdges(merge: (ED, ED) ⇒ ED): Graph[VD, ED]

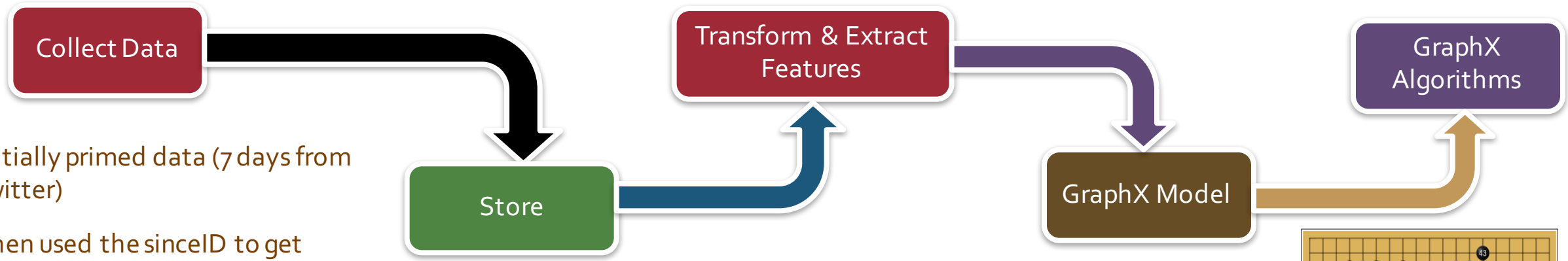
Merges multiple edges between two vertices into a single edge. For correct results, the graph must have been partitioned using partitionBy.



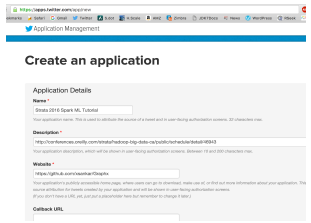
Edge Cut Vertex Cut

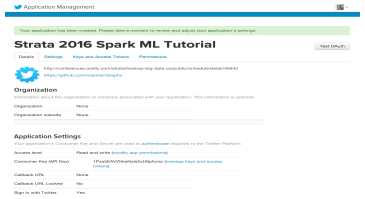
http://www.istc-cc.cmu.edu/publications/papers/2013/grades-graphx_with_fonts.pdf,https://www.sics.se/~amir/files/download/papers/jabeja-vc.pdf

AlphaGo Tweets Analytics Pipeline

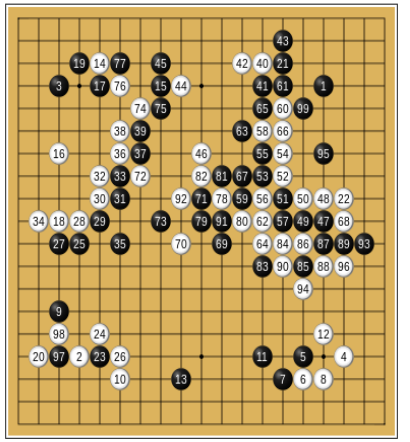


- Initially primed data (7 days from twitter)
- Then used the sinceID to get incremental tweets
- *Use application authentication for higher rate*
- *Used tweepy, wait_on_rate_limit=True, wait_on_rate_limit_notify=True*





- ~330K tweets (see Download program), 2GB
- => MongoDB ~820 MB w/compression)
- Rich data (*see MongoDB*)
  - Retweet, user ids, hash tags, locations et al
- This exercise only covers the retweet interest network



1-Extract Tweets

```
%pyspark
import tweepy

apiKey = "
apiSecret = "
accessToken = "
accessTokenSecret = "

auth = tweepy.AppAuthHandler(apiKey, apiSecret) # OAuthHandler(apiKey, apiSecret)
# auth.set_access_token(accessToken, accessTokenSecret)
api = tweepy.API(auth,wait_on_rate_limit=True,wait_on_rate_limit_notify=True)

if (not api):
    print ("Can't Authenticate")
    sys.exit(-1)

# http://www.karambelkar.info/2015/01/how-to-use-twitters-search-rest-api-most-effectively./
```

```
Downloading max 10000000 tweets
Downloaded 100 tweets
Downloaded 200 tweets
Downloaded 300 tweets
Downloaded 400 tweets
Downloaded 500 tweets
Downloaded 600 tweets
Downloaded 700 tweets
Downloaded 800 tweets
Downloaded 900 tweets
Downloaded 1000 tweets
Downloaded 1100 tweets
Downloaded 1200 tweets
Downloaded 1300 tweets
Downloaded 4374 tweets
Downloaded 4474 tweets
Downloaded 4574 tweets
Downloaded 4674 tweets
Downloaded 4774 tweets
Downloaded 4874 tweets
Downloaded 4974 tweets
Downloaded 5074 tweets
Downloaded 5166 tweets
No more tweets found
Downloaded 5166 tweets, Saved to tweets.txt
Took 80 seconds (outdated)
```

FINISHED

```
%pyspark
import sys
import jsonpickle
import os

searchQuery = 'AlphaGo'  # this is what we're searching for
maxTweets = 10000000 # Some arbitrary large number
tweetsPerQry = 100  # this is the max the API permits
fName = 'tweets.txt' # We'll store the tweets in a text file.


# If results from a specific ID onwards are reqd, set since_id to that ID.
# else default to no lower limit, go as far back as API allows
sinceId = 712410062820511744 #  711720090954108928 # 710883718903140353 # 710312094227300352 # 709817136437403649# 709550216467185664 # 709211104719872000L# None

# If results only below a specific ID are, set max_id to that ID.
# else default to no upper limit, start from the most recent tweet matching the search query.
max_id = -1L

tweetCount = 0
print("Downloading max {0} tweets".format(maxTweets))
with open(fName, 'w') as f:
    while tweetCount < maxTweets:
        try:
            if (max_id <= 0):
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry)
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,since_id=sinceId)
            else:
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,max_id=str(max_id - 1))
```

2-Pipeline screen shots

- Get max tweet id
- db.alphago.find().sort({id:-1}).limit(1).pretty()
  - "id" : NumberLong("7095502164671856 64"),
- db.alphago.find().sort({id:+1}).limit(1).pretty()
  - "id" : NumberLong("7092111324987146 27")
- /usr/local/mongo/bin/mongoimport --db admin --collection alphago --drop --file tweets-20160320.txt
  - 232296
  - Min : "id" : NumberLong("705845567537221632"),
  - Max : "id" : NumberLong("709211104719872000"),
- /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160314.txt
  - +21368
  - Min : "id" : NumberLong("705845567537221632"),
  - Max : "id" : NumberLong("709550216467185664"),
  - Count : 253664
- /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160315.txt
  - +38452
  - Min: "id_str" : "705845567537221632",
  - Max: "id" : NumberLong("709817136437403649")
  - Count : 292116

```
USS-Defiant:Strata-2016 ksankar$ /usr/local/mongo/bin/mongoexport --db admin --collection alphago --fieldFile export_fields.txt -o data-01.json
2016-03-13T23:07:30.219-0700    connected to: localhost
2016-03-13T23:07:31.220-0700    [........................]  admin.alphago  0/232296  (0.0%)
2016-03-13T23:07:32.220-0700    [........................]  admin.alphago  0/232296  (0.0%)
2016-03-13T23:07:33.221-0700    [........................]  admin.alphago  0/232296  (0.0%)
2016-03-13T23:07:34.221-0700    [........................]  admin.alphago  0/232296  (0.0%)
2016-03-13T23:07:35.220-0700    [........................]  admin.alphago  0/232296  (0.0%)
2016-03-13T23:07:36.219-0700    [........................]  admin.alphago  8000/232296  (3.4%)
2016-03-13T23:07:37.221-0700    [........................]  admin.alphago  8000/232296  (3.4%)
2016-03-13T23:07:38.220-0700    [........................]  admin.alphago  8000/232296  (3.4%)
2016-03-13T23:07:39.221-0700    [#.......................]  admin.alphago  16000/232296  (6.9%)
2016-03-13T23:07:40.221-0700    [#.......................]  admin.alphago  16000/232296  (6.9%)
2016-03-13T23:07:41.222-0700    [#.......................]  admin.alphago  16000/232296  (6.9%)
2016-03-13T23:07:42.221-0700    [#.......................]  admin.alphago  16000/232296  (6.9%)
2016-03-13T23:07:43.224-0700    [##......................]  admin.alphago  24000/232296  (10.3%)
2016-03-13T23:07:44.222-0700    [##......................]  admin.alphago  24000/232296  (10.3%)
2016-03-13T23:07:45.219-0700    [##......................]  admin.alphago  24000/232296  (10.3%)
2016-03-13T23:07:46.221-0700    [###.....................]  admin.alphago  32000/232296  (13.8%)
2016-03-13T23:07:47.219-0700    [###.....................]  admin.alphago  32000/232296  (13.8%)
2016-03-13T23:09:04.219-0700    [###################.....]  admin.alphago  184000/232296  (79.2%)
2016-03-13T23:09:05.221-0700    [###################.....]  admin.alphago  192000/232296  (82.7%)
2016-03-13T23:09:06.222-0700    [###################.....]  admin.alphago  192000/232296  (82.7%)
2016-03-13T23:09:07.220-0700    [###################.....]  admin.alphago  192000/232296  (82.7%)
2016-03-13T23:09:08.220-0700    [###################.....]  admin.alphago  192000/232296  (82.7%)
2016-03-13T23:09:09.222-0700    [####################....]  admin.alphago  200000/232296  (86.1%)
2016-03-13T23:09:10.224-0700    [####################....]  admin.alphago  200000/232296  (86.1%)
2016-03-13T23:09:11.222-0700    [####################....]  admin.alphago  200000/232296  (86.1%)
2016-03-13T23:09:12.221-0700    [####################...]  admin.alphago  208000/232296  (89.5%)
2016-03-13T23:09:13.223-0700    [####################...]  admin.alphago  208000/232296  (89.5%)
2016-03-13T23:09:14.221-0700    [####################...]  admin.alphago  208000/232296  (89.5%)
2016-03-13T23:09:15.220-0700    [####################...]  admin.alphago  208000/232296  (89.5%)
2016-03-13T23:09:16.219-0700    [####################...]  admin.alphago  208000/232296  (89.5%)
2016-03-13T23:09:17.221-0700    [#####################..]  admin.alphago  216000/232296  (93.0%)
2016-03-13T23:09:18.219-0700    [#####################..]  admin.alphago  216000/232296  (93.0%)
2016-03-13T23:09:19.221-0700    [#####################..]  admin.alphago  216000/232296  (93.0%)
2016-03-13T23:09:20.219-0700    [#####################..]  admin.alphago  224000/232296  (96.4%)
2016-03-13T23:09:21.223-0700    [#####################.]  admin.alphago  224000/232296  (96.4%)
2016-03-13T23:09:22.221-0700    [#####################.]  admin.alphago  224000/232296  (96.4%)
2016-03-13T23:09:23.221-0700    [#####################.]  admin.alphago  224000/232296  (96.4%)
2016-03-13T23:09:23.446-0700    [######################]  admin.alphago  232296/232296  (100.0%)
2016-03-13T23:09:23.446-0700    exported 232296 records
```

- /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160316.txt
  - +17331
  - Min: "id" : NumberLong("705845567537221632"),
  - Max: "id" : NumberLong("710312094227300352"),
  - Count : 309447
- /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160318.txt
  - +10797
  - Min: "id" : NumberLong("705845567537221632"),
  - Max: "id" : NumberLong("710883718903140353"),
  - Count : 320244
- /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160320.txt
  - +11511
  - Min: "id" : NumberLong("705845567537221632"),
  - Max: "id" : NumberLong("711720090954108928"),
  - Count : 331755
- /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160322.txt
  - +5166
  - Min: "id" : NumberLong("705845567537221632"),
  - Max: "id" : NumberLong("712410062820511744"),
  - Count : 336921

3-Twitter gives lots of fields in Mongo

```
> db.alphago.find().sort({id:-1}).limit(1).pretty()
{
        "_id" : ObjectId("56f1cc2a6e19ba0bcd5f9ddc"),
        "contributors" : null,
        "truncated" : false,
        "text" : "RT @cdixon: \"The human mind is still 50,000 times more energy efficient than machine intelligence.\" https://t.co/bChxqspT3M https://t
.co/pp…",
        "is_quote_status" : false,
        "in_reply_to_status_id" : null,
        "in_reply_to_user_id" : null,
        "id" : NumberLong("712410062820511744"),
        "favorite_count" : 0,
        "source" : "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitte
        "retweeted" : false,
        "coordinates" : null,
        "entities" : {
                "symbols" : [ ],
                "media" : [
                        {
                                "source_user_id" : 2529971,
                                "source_status_id_str" : "711298715646955520",
                                "expanded_url" : "http://twitter.com/cdixon/status/71129871
                                "display_url" : "pic.twitter.com/pp5ZtNQM0U",
                                "source_status_id" : NumberLong("711298715646955520"),
                                "media_url_https" : "https://pbs.twimg.com/media/Cd8KaKEVIA
                                "source_user_id_str" : "2529971",
                                "url" : "https://t.co/pp5ZtNQM0U",
                                "id_str" : "711298710643220480",
                                "sizes" : {
                                        "large" : {
                                                "h" : 669,
                                                "w" : 1024,
                                                "resize" : "fit"
                                        },
                                        "small" : {
                                                "h" : 222,
                                                "w" : 340,
                                                "resize" : "fit"
                                        }
                                        "resize" : "fit"
                                },
                                "medium" : {
                                        "h" : 392,
                                        "w" : 600,
                                        "resize" : "fit"
                                },
                                "thumb" : {
                                        "h" : 150,
                                        "w" : 150,
                                        "resize" : "crop"
                                }
                        },
                        "indices" : [
                                124,
                                140
                        ],
                        "type" : "photo",
                        "id" : NumberLong("711298710643220480"),
                        "media_url" : "http://pbs.twimg.com/media/Cd8KaKEVIAAw6mg.jpg"
                }
        ],
        "hashtags" : [ ],
        "user_mentions" : [
                {
                        "indices" : [
                                3,
                                10
                        ],
                        "screen_name" : "cdixon",
                        "id" : 2529971,
                        "name" : "Chris Dixon",
                        "id_str" : "2529971"
                }
        ],
        "urls" : [
                {
                        "indices" : [
                                100,
                                123
                        ],
                        "url" : "https://t.co/bChxqspT3M",
                        "expanded_url" : "http://jacquesmattheij.com/another-way-of-lo
        "favorited" : false,
        "user" : {
                "follow_request_sent" : null,
                "has_extended_profile" : false,
                "profile_use_background_image" : true,
                "id" : 2529971,
                "verified" : true,
                "profile_text_color" : "333333",
                "profile_image_url_https" : "https://pbs.twimg.com/profile_images/683496924104658944/8Oa5XAso_normal.png",
                "profile_sidebar_fill_color" : "DDEEF6",
                "entities" : {
                        "url" : {
                                "urls" : [
                                        {
                                                "indices" : [
                                                        0,
                                                        22
                                                ],
                                                "url" : "http://t.co/YviCcFNOiV",
                                                "expanded_url" : "http://cdixon.org/aboutme/",
                                                "display_url" : "cdixon.org/aboutme/"
                                        }
                                ]
                        },
                        "description" : {
                                "urls" : [ ]
                        }
                },
                "followers_count" : 218312,
                "protected" : false,
                "location" : "CA & NYC",
                "default_profile_image" : false,
                "id_str" : "2529971",
                "lang" : "en",
                "utc_offset" : -14400,
                "statuses_count" : 9225,
                "description" : "programming, philosophy, history, internet, startups, investing",
                "friends_count" : 3376,
                "profile_background_image_url_https" : "https://pbs.twimg.com/profile_background_images/543964120664403968/0IKm7siW.png",
                "profile_link_color" : "89C9FA",
```

4-Extract Retweet Fields & 5-Store in csv

FINISHED

```
%pyspark
import pymongo
from pymongo import MongoClient

print "     pymongo : %s" % (pymongo.version)
client=MongoClient('localhost', 27017)
db = client["admin"] #db name
alphago=db["alphago"]

cursor = alphago.find({},{"id":1,"text":1,"user.id":1,"user.followers_count":1,"user.friends_count":1,"user.name":1,"user.time_zone":1,"user.location":1,
    "retweet_count":1,"retweeted_status.user.id":1,"retweeted_status.user.followers_count":1,"retweeted_status.user.friends_count":1,"retweeted_status.user.name":1,
    "retweeted_status.user.time_zone":1,"retweeted_status.user.location":1})
print cursor.count()
```

```
%pyspark
# write csv file
count = 0
#with open('reTweetNetwork-small.psv', 'w') as f:
with open('reTweetNetwork-large.psv', 'w') as f:
    for doc in cursor:
        tweetId = doc["id"]
        tweetCount = doc["retweet_count"]
        tweetText = "NA" #doc["text"].encode('ascii', 'ignore').replace('\n', ' ').replace('\r', '')
        srcUserId = doc["user"]["id"]
        srcUserName = doc["user"]["name"].encode('ascii', 'ignore')
        if (srcUserName is not None):
           srcUserName = srcUserName.rstrip().lstrip()
        if (srcUserName is None):
            srcUserName="NA"
        srcUserLocation = doc["user"]["location"].encode('ascii', 'ignore')
        if (srcUserLocation is not None):
           srcUserLocation = srcUserLocation.rstrip().lstrip()
        if (srcUserLocation is None):
            srcUserLocation ="NA"
        srcUserTz = doc["user"]["time_zone"]
        if (srcUserTz is not None):
           srcUserTz = srcUserTz.rstrip().lstrip()
        if (srcUserTz is None):
            srcUserTz="NA"
        srcUserFr = doc["user"]["friends_count"]
        if (not (type( srcUserFr) is int)):
            srcUserFr = 0
        srcUserFol = doc["user"]["followers_count"]
        if (not (type( srcUserFol) is int)):
            srcUserFol = 0
        if ( "retweeted_status" in doc):
            destUserId = doc["retweeted_status"]["user"]["id"]
            destUserName = doc["retweeted_status"]["user"]["name"].encode('ascii', 'ignore')
            if (destUserName is not None):
               destUserName = destUserName.rstrip().lstrip()
            if (destUserName is None):
                destUserName="NA"
            destUserLocation = doc["retweeted_status"]["user"]["location"].encode('ascii', 'ignore')
            if (destUserLocation is not None):
               destUserLocation = destUserLocation.rstrip().lstrip()
            if (destUserLocation is None):
                destUserLocation ="NA"
            destUserTz = doc["retweeted_status"]["user"]["time_zone"]
            if (destUserTz is not None):
               destUserTz = destUserTz.rstrip().lstrip()
            if (destUserTz is None):
                destUserTz = "NA"
            destUserFr = doc["retweeted_status"]["user"]["friends_count"]
            destUserFol = doc["retweeted_status"]["user"]["followers_count"]
        else:
            destUserId = "NA"
            destUserName = "NA"
            destUserLocation = "NA"
            destUserTz =  "NA"
            destUserFr =  "NA"
            destUserFol = "NA"
        if ("retweeted_status" in doc):  # write only retweets, as we want to create the retwwet netwoek
            f.write('{0}|{1}|"{2}"|{3}|"{4}"|"{5}"|"{6}"|{7}|{8}|{9}|"{10}"|"{11}"|"{12}"|{13}|{14}\n'.format(tweetId, tweetCount, tweetText, srcUserId, srcUserName,
                srcUserLocation, srcUserTz, srcUserFr, srcUserFol, destUserId, destUserName, destUserLocation, destUserTz, destUserFr, destUserFol))
            # count += 1
            #if (count > 10000):
            #    break
print "** Done **"
```

6-Read as dataframe
7-Create Vertices, Edges & Objects
8-Finally the graph & run algorithms

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val df = sqlContext.read
    .format("com.databricks.spark.csv")
    .option("header", "false") // Use first line of all files as header
    .option("inferSchema", "true") // Automatically infer data types
    .option("delimiter","|")
    .load("reTweetNetwork-small.psv")
df.show(5)
df.count()
```

```
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@...
df: org.apache.spark.sql.DataFrame = [C0: bigint, C1: int, C2: string, C3: bigint, C4: string, C5: string, C6: string, C7: int, C8: int, C9: bigint, C10: string, C11:
string, C12: string, C13: int, C14: int]
+------------------+---+---+----------+--------------+--------------------+--------------------+----+----+--------+--------------+--------------------+-------------
------+----+-------+
|               C0| C1| C2|        C3|            C4|                  C5|                  C6|  C7|  C8|      C9|           C10|                 C11|
C12| C13|    C14|
+------------------+---+---+----------+--------------+--------------------+--------------------+----+----+--------+--------------+--------------------+-------------
------+----+-------+
|709210987015114752|  2| NA| 589277295| Unsigned Heroes|          Seattle WA|             Arizona| 919|2703|12798452|Erick Schonfeld|            New York|Central Time
(US ...|1292|  77657|
|709210994875039744| 25| NA| 271007340|    Johnny Apuan|                    |              None|1252|  99| 1344951|          WIRED|San Francisco/New...|Pacific Time
(US ...| 264|5714763|
|709210924771467264| 39| NA| 166001040|         Nedpool|Canterbury New Ze...|          Wellington|1033| 572| 8917142|   Dan Kaminsky|Chief Scientist, ...|Pacific Time
(US ...| 401|  50229|
|709210923928559616| 25| NA|3244208187|Deen over Duniya|                Mars|              None|  93| 223| 1344951|          WIRED|San Francisco/New...|Pacific Time
(US ...| 264|5714763|
|709210919729963008| 25| NA|2749754251|    Farzan Sabet|                    |Eastern Time (US ...| 373| 862| 1344951|          WIRED|San Francisco/New...|Pacific Time
(US ...| 264|5714763|
+------------------+---+---+----------+--------------+--------------------+--------------------+----+----+--------+--------------+--------------------+-------------
------+----+-------+
only showing top 5 rows
```

```
// Create vertices & edges
case class User(name:String, location:String, tz : String, fr:Int,fol:Int)
case class Tweet(id:String,count:Int)

val graphData = df.rdd
graphData.take(2)

val vert1 = graphData.map(row => (row(3).toString.toLong,User(row(4).toString,row(5).toString,row(6).toString,row(7).toString.toInt,row(8).toString.toInt)))
vert1.count()
vert1.take(3)

val vert2 = graphData.map(row => (row(9).toString.toLong,User(row(10).toString,row(11).toString,row(12).toString,row(13).toString.toInt,row(14).toString.toInt)))
vert2.count()
vert2.take(3)

val vertX = vert1.++(vert2)
vertX.count()

val edgX = graphData.map(row => (Edge(row(3).toString.toLong,row(9).toString.toLong,(row(0),row(1)))))
edgX.take(3)

val rtGraph = Graph(vertX,edgX)
```

```
val ranks = rtGraph.pageRank(0.1).vertices
ranks.take(2)
val topUsers = ranks.sortBy(_._2,false).take(3).foreach(println)
val topUsersWNames = ranks.join(rtGraph.vertices).sortBy(_._2._1,false).take(3).foreach(println)
```

```
ranks: org.apache.spark.graphx.VertexRDD[Double] = VertexRDDImpl[627] at RDD at VertexRDD.scala:57
res92: Array[(org.apache.spark.graphx.VertexId, Double)] = Array((144366820,0.15), (9734132,0.15))
(11821362,322.9447926136434)
(1482581556,84.10384874154467)
(14497118,43.956874999999826)
topUsers: Unit = ()
(11821362,(322.9447926136434,User(Statsman Bruno,Stockholm,Stockholm,53,79094)))
(1482581556,(84.10384874154467,User(Demis Hassabis,,None,11,22637)))
(14497118,(43.956874999999826,User(Ken Kawamoto,Kokubunji, Tokyo,Tokyo,549,5675)))
topUsersWNames: Unit = ()
```

Took 3 seconds

```
%md
### How many retweets ?
```

How many retweets ?

Took 0 seconds

```
val iDeg = rtGraph.inDegrees
val oDeg = rtGraph.outDegrees
```

```
iDeg: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[648] at RDD at VertexRDD.scala:57
oDeg: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[475] at RDD at VertexRDD.scala:57
```

Took 1 seconds

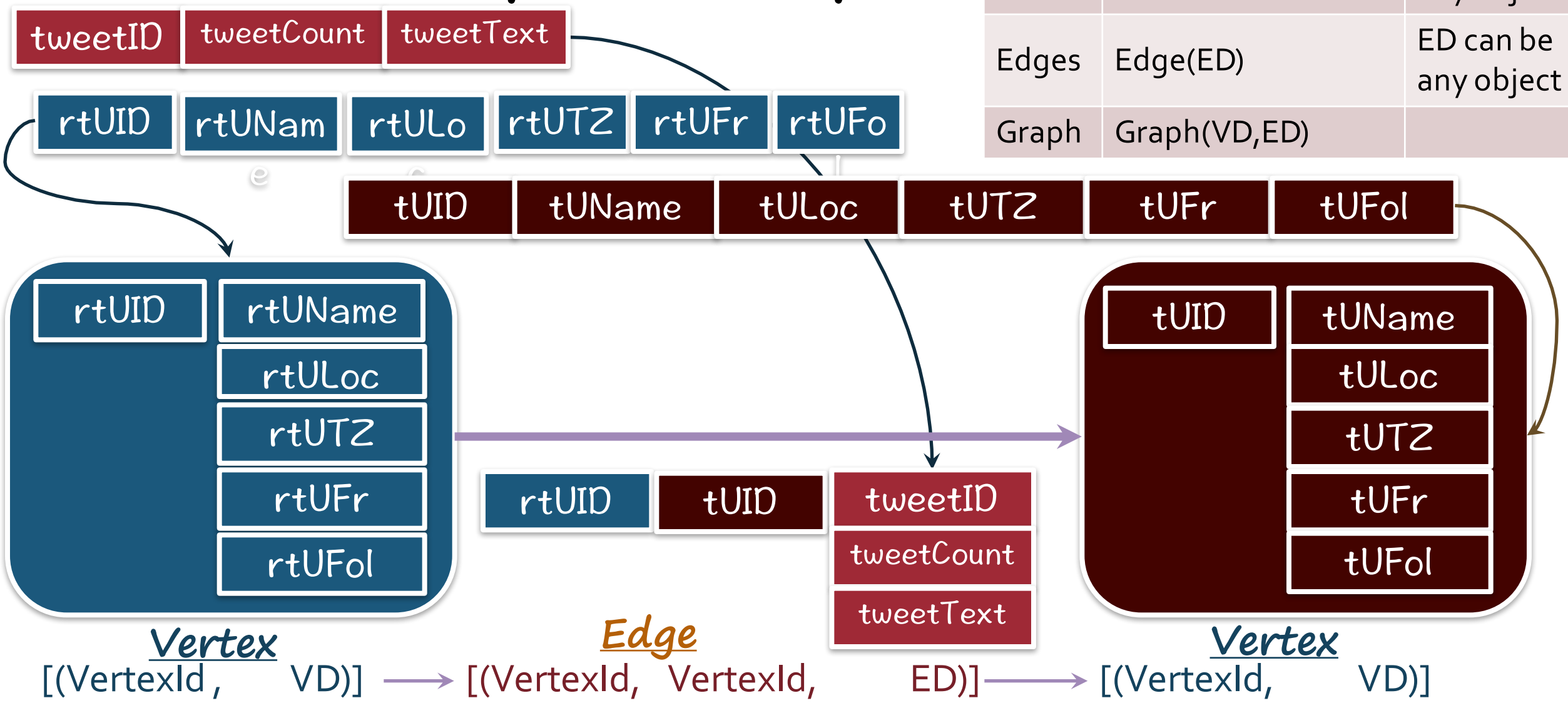
```
iDeg.take(3)
iDeg.sortBy(_._2,false).take(3).foreach(println)
iDeg
```

```
res36: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((630222308,1), (149925820,11), (169630984,2))
res37: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((11821362,2574), (1482581556,860), (14497118,360))
```

Took 1 seconds (outdated)

The Art of an AlphaGo GraphX

Vertex	Vertex(VertexId,VD)	VD can be any object
Edges	Edge(ED)	ED can be any object
Graph	Graph(VD,ED)	



```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val df = sqlContext.read
    .format("com.databricks.spark.csv")
    .option("header", "false") // Use first line of all files as header
    .option("inferSchema", "true") // Automatically infer data types
    .option("delimiter","|")
    .load("reTweetNetwork-small.psv")
df.show(5)
df.count()
```

1

```
// Create vertices & edges
case class User(name:String, location:String, tz : String, fr:Int, fol:Int)
case class Tweet(id:String, count:Int)

val graphData = df.rdd
//graphData.take(2).foreach(println)

val vert1 = graphData.map(row => (row(3).toString.toLong, User(row(4).toString,row(5).toString,row(6).toString,row(7).toString.toInt,row(8).toString.toInt)))
vert1.count()
//vert1.take(3).foreach(println)

val vert2 = graphData.map(row => (row(9).toString.toLong, User(row(10).toString,row(11).toString,row(12).toString,row(13).toString.toInt,row(14).toString.toInt)))
vert2.count()
//vert2.take(3).foreach(println)

val vertX = vert1.++(vert2)
vertX.count()
vertX.take(3).foreach(println)
```

2

```
defined class User
defined class Tweet
graphData: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[962] at rdd at <console>:38
vert1: org.apache.spark.rdd.RDD[(Long, User)] = MapPartitionsRDD[963] at map at <console>:43
res85: Long = 10001
vert2: org.apache.spark.rdd.RDD[(Long, User)] = MapPartitionsRDD[964] at map at <console>:43
res88: Long = 10001
vertX: org.apache.spark.rdd.RDD[(Long, User)] = UnionRDD[965] at $plus$plus at <console>:47
res91: Long = 20002
(589277295,User(Unsigned Heroes,Seattle WA,Arizona,919,2703))
(271007340,User(Johnny Apuan,NA,NA,1252,99))
(166001040,User(Nedpool,Canterbury New Zealand,Wellington,1033,572))
```

```
val edgX = graphData.map(row => (Edge(row(3).toString.toLong, row(9).toString.toLong, Tweet(row(0).toString,row(1).toString.toInt))))
edgX.take(3).foreach(println)
edgX.count()
```

```
edgX: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Tweet]] = MapPartitionsRDD[966] at map at <console>:41
Edge(589277295,12798452,Tweet(709210987015114752,2))
Edge(271007340,1344951,Tweet(709210994875039744,25))
Edge(166001040,8917142,Tweet(709210924771467264,39))
res95: Long = 10001
```

3

```
val rtGraph = Graph(vertX,edgX)
//How big ?
rtGraph.vertices.count
rtGraph.edges.count
```

4

```
rtGraph: org.apache.spark.graphx.Graph[User,Tweet] = org.apache.spark.graphx.impl.GraphImpl@244747b4
res98: Long = 9743
res99: Long = 10001
```

```
val ranks = rtGraph.pageRank(0.1).vertices
ranks.take(2)
val topUsers = ranks.sortBy(_._2,false).take(3).foreach(println)
val topUsersWNames = ranks.join(rtGraph.vertices).sortBy(_._2._1,false).take(3).foreach(println)
```

```
ranks: org.apache.spark.graphx.VertexRDD[Double] = VertexRDDImpl[1136] at RDD at VertexRDD.scala:57
res101: Array[(org.apache.spark.graphx.VertexId, Double)] = Array((144366820,0.15), (9734132,0.15))
(11821362,322.9447926136434)
(1482581556,84.29509874154466)
(14497118,43.956874999999826)
topUsers: Unit = ()
(11821362,(322.9447926136434,User(Statsman Bruno,Stockholm,Stockholm,53,79094)))
(1482581556,(84.29509874154466,User(Demis Hassabis,NA,NA,11,22637)))
(14497118,(43.956874999999826,User(Ken Kawamoto,Kokubunji, Tokyo,Tokyo,549,5675)))
topUsersWNames: Unit = ()
Took 3 seconds
```

5

I enjoyed a lot preparing the materials ...
Hope you enjoyed more attending ...
@ksankar