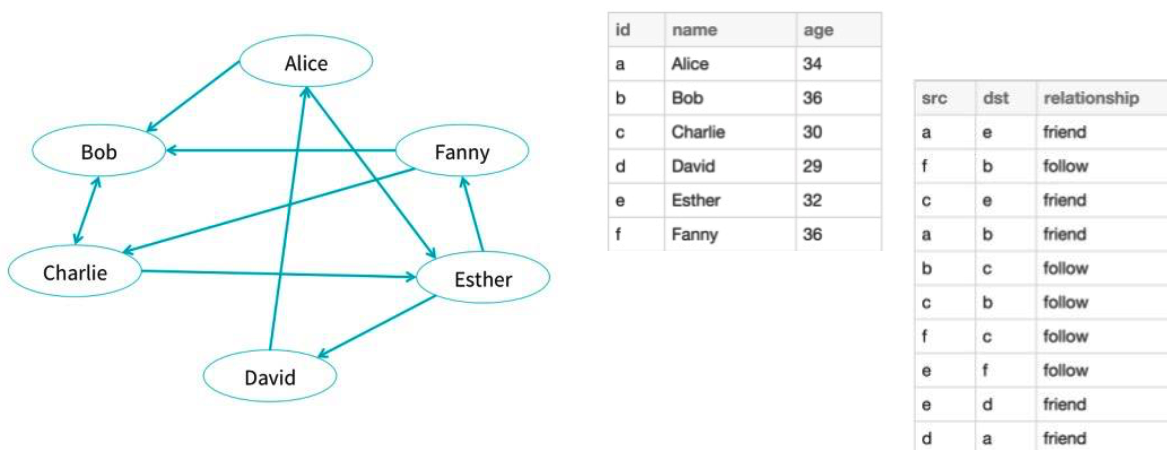
**1. Input**



# Use the following command to start certain version of spark python

# Spark version usually is updated fast

# Fix the version for each project to make sure repeatability

>>> pyspark --packages graphframes:graphframes:0.1.0-spark1.6

# Spark generated a context. Just to check the version here.

>>> sc.version

# In practice, we should only import the packages needed in project.

# Here import everything for convenience.

>>> from graphframes import \*

# Input data using Spark Dataframe

# Define vertices and edges of a graph

>>> v = sqlContext.createDataFrame([   
("a", "Alice", 34),   
("b", "Bob", 36),   
("c", "Charlie", 30),   
("d", "David", 29),   
("e", "Esther", 32),   
("f", "Fanny", 36)   
], ["id", "name", "age"])   
  
>>> e = sqlContext.createDataFrame([   
("a", "e", "friend"),   
("f", "b", "follow"),   
("c", "e", "friend"),   
("a", "b", "friend"),   
("b", "c", "follow"),   
("c", "b", "follow"),   
("f", "c", "follow"),   
("e", "f", "follow"),   
("e", "d", "friend"),   
("d", "a", "friend")   
], ["src", "dst", "relationship"])

# Define graph frame

>>> gr = GraphFrame(v, e)

>>> print gr

GraphFrame(v:[id: string, name: string, age: bigint], e:[src: string, dst: string, relationship: string])

# Show features of existing graph

>>> gr.edges.show()

>>> gr.vertices.show()

>>> gr.inDegrees.show()

**2. Generate sub-graphs**

>>> gr.vertices.filter("age > 25").show()

+---+-------+---+

| id| name|age|

+---+-------+---+

| a| Alice| 34|

| b| Bob| 36|

| c|Charlie| 30|

| d| David| 29|

| e| Esther| 32|

| f| Fanny| 36|

+---+-------+---+

>>> gr.inDegrees.show()

+---+--------+

| id|inDegree|

+---+--------+

| a| 1|

| b| 3|

| c| 2|

| d| 1|

| e| 2|

| f| 1|

+---+--------+

# Lazy computation: the command won’t activate any computation until it happens

>>> nodes = gr.outDegrees.filter("outDegree >= 2")

>>> nodes.show()

+---+---------+

| id|outDegree|

+---+---------+

| a| 2|

| c| 2|

| e| 2|

| f| 2|

+---+---------+

# Another example for sub-graph

>>> v2 = gr.vertices.filter("age > 30")

>>> e2 = gr.edges.filter("relationship = 'friend'")

>>> g2 = GraphFrame(v2, e2)

# Select sub-graph based on edges "e" of type "follow"

# pointing from a younger user "a" to an older user "b"

>>> paths = gr.find("(a)-[e]->(b)")\

.filter("e.relationship = 'follow'")\

.filter("a.age < b.age")

>>> paths.show()

+------------+--------------+------------+

| e| a| b|

+------------+--------------+------------+

|[c,b,follow]|[c,Charlie,30]| [b,Bob,36]|

|[e,f,follow]| [e,Esther,32]|[f,Fanny,36]|

+------------+--------------+------------+

# "paths" contains vertex info. Extract the edges.

>>> e2 = paths.select("e.src", "e.dst", "e.relationship")

>>> e2.show()

+---+---+------------+

|src|dst|relationship|

+---+---+------------+

| c| b| follow|

| e| f| follow|

+---+---+------------+

# In Spark 1.5+, the user may simplify this call:

# e2 = paths.select("e.\*")

**3. Find patterns**

# Find connected component

# Motif: A->B->C but not A->C

>>> results = gr.find("(A)-[]->(B); (B)-[]->(C); !(A)-[]->(C)")

>>> results # the results is shown to be Dataframe

DataFrame[A: struct<id:string,name:string,age:bigint>, B: struct<id:string,name:string,age:bigint>, C: struct<id:string,name:string,age:bigint>]

>>> results.show() # to check the results  
+--------------+--------------+--------------+

| A| B| C|

+--------------+--------------+--------------+

| [e,Esther,32]| [f,Fanny,36]|[c,Charlie,30]|

|[c,Charlie,30]| [b,Bob,36]|[c,Charlie,30]|

| [f,Fanny,36]|[c,Charlie,30]| [e,Esther,32]|

| [a,Alice,34]| [e,Esther,32]| [f,Fanny,36]|

| [a,Alice,34]| [b,Bob,36]|[c,Charlie,30]|

| [b,Bob,36]|[c,Charlie,30]| [b,Bob,36]|

| [e,Esther,32]| [d,David,29]| [a,Alice,34]|

| [a,Alice,34]| [e,Esther,32]| [d,David,29]|

| [b,Bob,36]|[c,Charlie,30]| [e,Esther,32]|

|[c,Charlie,30]| [e,Esther,32]| [f,Fanny,36]|

| [e,Esther,32]| [f,Fanny,36]| [b,Bob,36]|

|[c,Charlie,30]| [e,Esther,32]| [d,David,29]|

| [d,David,29]| [a,Alice,34]| [b,Bob,36]|

| [d,David,29]| [a,Alice,34]| [e,Esther,32]|

+--------------+--------------+--------------+

>>> results = gr.find("(A)-[]->(B); (B)-[]->(C); !(A)-[]->(C); !(C)-[]->(A) ")

>>> results.show()

+--------------+--------------+--------------+

| A| B| C|

+--------------+--------------+--------------+

|[c,Charlie,30]| [b,Bob,36]|[c,Chaßlie,30]|

| [a,Alice,34]| [e,Esther,32]| [f,Fanny,36]|

| [a,Alice,34]| [b,Bob,36]|[c,Charlie,30]|

| [b,Bob,36]|[c,Charlie,30]| [b,Bob,36]|

| [b,Bob,36]|[c,Charlie,30]| [e,Esther,32]|

| [e,Esther,32]| [f,Fanny,36]| [b,Bob,36]|

|[c,Charlie,30]| [e,Esther,32]| [d,David,29]|

| [d,David,29]| [a,Alice,34]| [b,Bob,36]|

+--------------+--------------+--------------+

# How to generate a graph without direction? Direction is self-defined.

# A sub-graph is generated here. Then computation can be made in the sub-graph.

>>> edges2 = gr.edges.filter("relationship = 'friend'")

>>> gr2 = GraphFrame(v,edges2)

>>> gr2.edges.show()

+---+---+------------+

|src|dst|relationship|

+---+---+------------+

| a| e| friend|

| c| e| friend|

| a| b| friend|

| e| d| friend|

| d| a| friend|

+---+---+------------+

**4. Connected components and shortest path**

>>> concom = gr.connectedComponents()

>>> concom.select("id", "component").orderBy("component").show()

+---+---------+

| id|component|

+---+---------+

| a| 1|

| c| 1|

| d| 1|

| e| 1|

| b| 1|

| f| 1|

+---+---------+

>>> concom2 = gr.stronglyConnectedComponents(maxIter=6)

>>> concom2.select("id", "component").orderBy("component").show()

+---+---------+

| id|component|

+---+---------+

| b| 1|

| a| 1|

| c| 1|

| d| 1|

| f| 1|

| e| 1|

+---+---------+

# Shortest path

>>> results = g.shortestPaths(landmarks=["a", "d"])

>>> results.select("id", "distances").show()

+---+-------------------+

| id| distances|

+---+-------------------+

| a|Map(a -> 0, d -> 2)|

| b|Map(d -> 3, a -> 4)|

| c|Map(d -> 2, a -> 3)|

| d|Map(d -> 0, a -> 1)|

| e|Map(d -> 1, a -> 2)|

| f|Map(d -> 3, a -> 4)|

+---+-------------------+

**5. PageRank**

# In PageRank algorithm, randomly assign a probability at the beginning

>>> pk = gr.pageRank(resetProbability=0.01, maxIter=10)

>>> print pk

GraphFrame(v:[id: string, name: string, age: bigint, pagerank: double], e:[src: string, dst: string, relationship: string, weight: double])

>>> pk.vertices.select("id","name","pagerank").show()

+---+-------+-------------------+

| id| name| pagerank|

+---+-------+-------------------+

| a| Alice|0.06733726684755689|

| b| Bob| 0.1462404244456829|

| c|Charlie|0.17065485805878866|

| d| David|0.06308306703688421|

| e| Esther|0.11757179102190443|

| f| Fanny|0.06308306703688421|

+---+-------+-------------------+

**6. Save graphs**

# Save edges and vertices separately as Parquet to some locations

>>> gr.vertices.write.parquet("hdfs:///user/aisjiaw/tmp/vertices")

# Check if it is saved correctly in cluster

>>> hdfs dfs -ls /user/aisjiaw/tmp/vertices

>>> gr.edges.write.parquet("hdfs:///user/aisjiaw/tmp/edges")

# Read the vertices back

>>> sameV = sqlContext.read.parquet("hdfs:///user/aisjiaw/tmp/vertices")

>>> sameV.show()

+---+-------+---+

| id| name|age|

+---+-------+---+

| a| Alice| 34|

| b| Bob| 36|

| c|Charlie| 30|

| d| David| 29|

| e| Esther| 32|

| f| Fanny| 36|

+---+-------+---+

>>> sameG = sqlContext.read.parquet("hdfs:///user/aisjiaw/tmp/edges")

# Create an identical GraphFrame

>>> sameG = GraphFrame(sameV, sameE)