МИРЭА
Российский технологический университет

Big Data

Лебедев Артём Сергеевич

Учебный год 2021/2022

Лекция 15

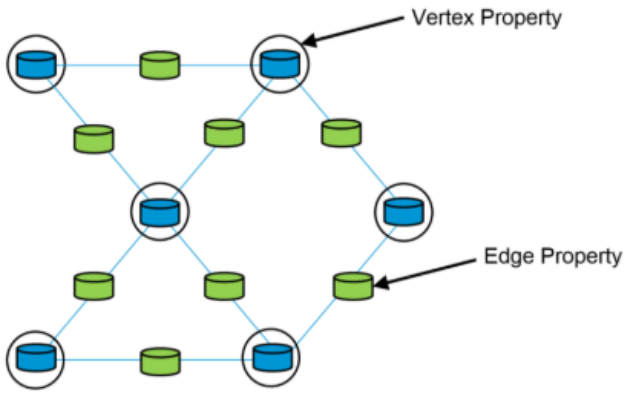
Обработка графов с применением GraphFrames

GraphFrames

- Пакет для Apache Spark, который предоставляет вычисления на графах на основе DataFrame.
  - Так же как и GraphX, функционирующий на основе RDD.
- API доступен из Scala, Java и Python.
- Призван обеспечить как функциональность GraphX, так и расширенную функциональность, используя преимущества Spark DataFrames:
  - поиск подграфов с заданными свойствами;
  - сериализация на основе DataFrame (сохранение и загрузка графов);
  - запросы к графам с высокой степенью выразительности.

Представление графов в GraphFrames

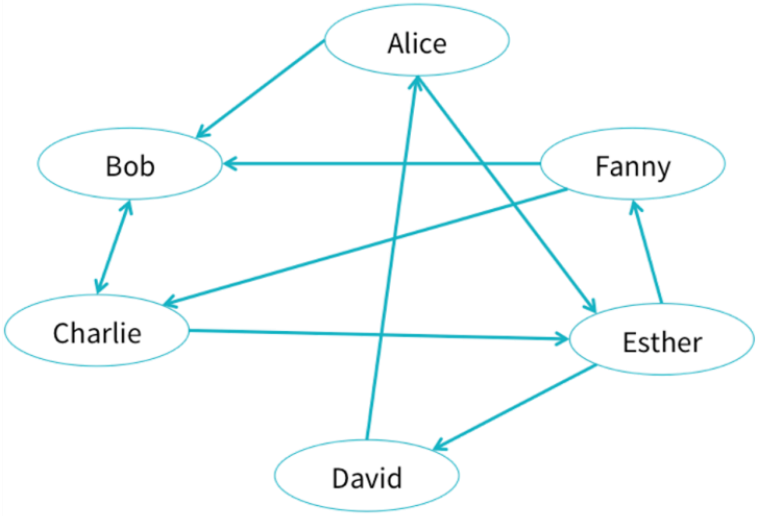
- **Vertex DataFrame:** должен содержать специальный столбец с именем «id», который определяет уникальные идентификаторы для каждой вершины в графе.
- **Edge DataFrame:** должен содержать два специальных столбца: «src» (идентификатор исходной вершины ребра) и «dst» (целевой идентификатор вершины ребра).
- Прочие столбцы могут представлять пользовательские атрибуты, ассоциированные с вершинами или ребрами.



Пример графа: социальная сеть

Vertex DataFrame

id	name	age
a	Alice	34
b	Bob	36
c	Charlie	30
d	David	29
e	Esther	32
f	Fanny	36



Edge DataFrame

src	dst	relationship
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

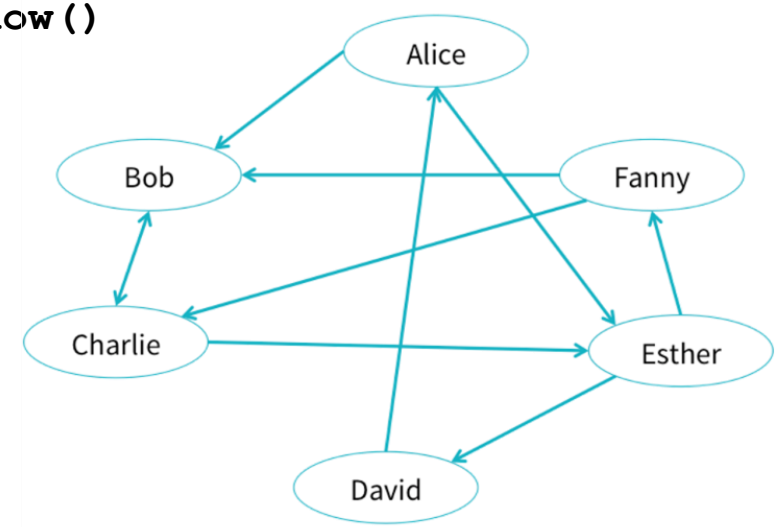
```
>>> v = spark.createDataFrame([("a", "Alice", 34), ("b",
"Bob", 36), ("c", "Charlie", 30), ("d", "David", 29), ("e",
"Esther", 32), ("f", "Fanny", 36)], ["id", "name", "age"])

>>> e = spark.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"])

>>> from graphframes import *
>>> g = GraphFrame(v, e)
```

Простые графовые запросы

```
# Сколько пользователей в нашей социальной сети старше 35 лет?
>>> g.vertices.filter("age > 35").show()
+---+-----+---+
| id| name|age|
+---+-----+---+
|  b|  Bob| 36|
|  f|Fanny| 36|
+---+-----+---+
# Каков возраст самого молодого участника?
>>> g.vertices.groupBy().min("age").show()
+--------+
|min(age)|
+--------+
|      29|
+--------+
# Сколько существует дружеских связей?
>>> g.edges.filter("relationship = 'follow'").count()
5
```

Простые графовые запросы: полустепень захода вершины

```
# Отобразить полустепень захода для каждой вершины
>>> g.inDegrees.show()
+---+--------+
| id|inDegree|
+---+--------+
|  f|       1|
|  e|       2|
|  d|       1|
|  c|       2|
|  b|       3|
|  a|       1|
+---+--------+
```




```
# У скольких пользователей есть как минимум 2 подписчика?
>>> g.inDegrees.filter("inDegree >= 2").count()
3
```

Поиск шаблонов с заданными свойствами. Предметно-ориентированный язык запросов

- Основная единица паттерна - ребро.
  - «`(a)-[e]->(b)`» выражает ребро `e` от вершины `a` до вершины `b`.
- Паттерн выражается как объединение ребер. Ребра можно объединять точками с запятой.
- Имена, присваиваемые вершинам и ребрам:
  - могут идентифицировать общие элементы среди ребер:
    - «`(a)-[e]->(b); (b)-[e2]->(c)`» определяет два ребра: от `a` до `b` и от `b` до `c`.
  - не обязательно идентифицируют различные элементы:
    - «`(a)-[e]->(b); (b)-[e2]->(c)`» позволяет `a` и `c` указывать на одну и ту же вершину.
  - используются как имена столбцов в результирующем DataFrame:
    - столбец будет представлять структуру с полями, соответствующими схеме Vertex или Edge DataFrame.

Поиск шаблонов с заданными свойствами. Предметно-ориентированный язык запросов

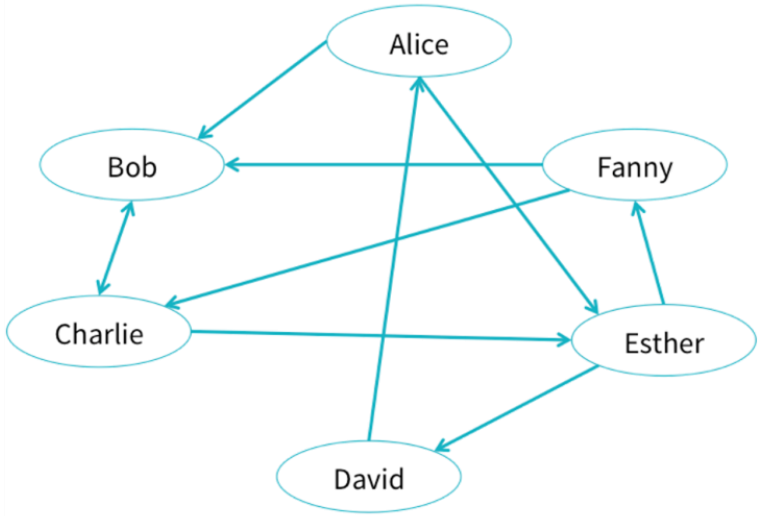
- Анонимные вершины и ребра:
  - Допускается опускать имена вершин или ребер, когда они не нужны.
    - «`(a)-[]->(b)`» выражает ребро между вершинами `a` и `b`, но не присваивает ему имя.
    - «`(a)-[e]->()`» указывает исходящее из вершины `a` ребро, но не указывает вершину назначения.
  - В результирующем DataFrame не будет столбца для анонимного элемента.
- Ребро может содержать отрицание, если оно не должно присутствовать в результате выполнения запроса.
  - «`(a)-[]->(b); !(b)-[]->(a)`» находит ребра от `a` до `b`, для которых нет ребра в обратном направлении.
- Ограничения:
  - Паттерны не могут содержать ребра без каких-либо именованных элементов:
    - «`()-[]->()`» и «`!()-[]->()`» являются некорректными.
  - Паттерны не могут содержать именованные ребра внутри отрицаний (поскольку эти именованные ребра никогда не появятся в результатах).
    - «`!(a)-[ab] ->(b)`» некорректно, но «`!(a)-[]->(b)`» допустимо.

Поиск шаблонов с заданными свойствами: простые примеры

```
# Найти обоюдные связи в социальной сети:
>>> r = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
>>> r.show()
+----------------+-------------+----------------+-------------+
|               a|            e|               b|           e2|
+----------------+-------------+----------------+-------------+
|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|[b, c, follow]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, b, follow]|
+----------------+-------------+----------------+-------------+
# Ограничить возраст второго участника:
>>> r.filter("b.age > 30").show()
+----------------+-------------+-----------+-------------+
|               a|            e|          b|           e2|
+----------------+-------------+-----------+-------------+
|[c, Charlie, 30]|[c, b, follow]|[b, Bob, 36]|[b, c, follow]|
+----------------+-------------+-----------+-------------+
```

Поиск шаблонов с заданными свойствами: рекомендация подписки

```
# Пусть A подписан на B, B подписан на C, но A не подписан на C:
>>> r = g.find("(A)-[]->(B); (B)-[]->(C); !(A)-[]->(C)")
# При этом A и C должны представлять разных участников:
>>> r = r.filter("A.id != C.id")
# Оставим в DataFrame только A и C, поскольку B не требуется:
>>> r = r.select("A", "C")
>>> r.show()
+---------------+---------------+
|              A|              C|
+---------------+---------------+
| [e, Esther, 32]|[c, Charlie, 30]|
| [e, Esther, 32]|  [a, Alice, 34]|
|[c, Charlie, 30]|  [f, Fanny, 36]|
|  [d, David, 29]|    [b, Bob, 36]|
|  [a, Alice, 34]|  [d, David, 29]|
|[c, Charlie, 30]|  [d, David, 29]|
|  [d, David, 29]| [e, Esther, 32]|
|  [f, Fanny, 36]| [e, Esther, 32]|
|  [a, Alice, 34]|  [f, Fanny, 36]|
| [e, Esther, 32]|    [b, Bob, 36]|
|  [a, Alice, 34]|[c, Charlie, 30]|
|    [b, Bob, 36]| [e, Esther, 32]|
+---------------+---------------+
```



Выборка подграфов

- filterVertices(condition) - фильтрует вершины на основе выражения, удаляет ребра, содержащие все отброшенные вершины.
- filterEdges(condition) - фильтрует ребра на основе выражения, сохраняя все вершины.
- dropIsolatedVertices() - отбрасывает изолированные вершины.

```
# Выбрать подграф с друзьями старше 30:
>>> sg = g.filterVertices("age > 30").filterEdges("relationship =
'friend'").dropIsolatedVertices()
```

```
>>> sg.vertices.show()
+---+------+---+
| id|  name|age|
+---+------+---+
|  e|Esther| 32|
|  b|   Bob| 36|
|  a| Alice| 34|
+---+------+---+
```

```
>>> sg.edges.show()
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  a|  e|      friend|
|  a|  b|      friend|
+---+---+------------+
```

Выборка подграфов с поиском шаблонов

```
# Шаблон, в котором связь "follow" направлена от младшего пользователя к старшему:
>>> r = g.find("(a)-[e]->(b)").filter("e.relationship = 'follow'").filter("a.age < b.age")
>>> r.show()
+----------------+-------------+-------------+
|               a|            e|            b|
+----------------+-------------+-------------+
| [e, Esther, 32]|[e, f, follow]|[f, Fanny, 36]|
|[c, Charlie, 30]|[c, b, follow]|  [b, Bob, 36]|
+----------------+-------------+-------------+
# Новый GraphFrame, представляющий подграф:
>>> g1 = GraphFrame(g.vertices, r.select("e.*"))
```

Алгоритмы на графах: параллельный поиск в ширину

- находит кратчайший путь (пути) от одной вершины (или набора вершин) до другой вершины (или набора вершин). Начальная и конечная вершины указываются как выражения Spark DataFrame.

```
# Кратчайший путь от Esther до пользователя моложе 32 лет вдоль ребер, отличных от «дружбы», с ограничением длины пути не больше 3:
g.bfs("name = 'Esther'", "age < 32", edgeFilter="relationship != 'friend'", maxPathLength=3).show()
+---------------+--------------+--------------+--------------+----------------+
|           from|            e0|            v1|            e1|              to|
+---------------+--------------+--------------+--------------+----------------+
|[e, Esther, 32]|[e, f, follow]|[f, Fanny, 36]|[f, c, follow]|[c, Charlie, 30]|
+---------------+--------------+--------------+--------------+----------------+
```

Алгоритмы на графах: PageRank

- Поддерживаемые варианты PageRank:
  - с фиксированным числом итераций:
    - указывается их количество `maxIter`.
  - до выполнения условия схождения алгоритма:
    - указывается порог точности `tol`.
  - персонализированный и неперсонализированный:
    - указывается вершина `sourceId` для персонализированного варианта.

```
# Запуск с фиксированным числом итераций:
>>> pr = g.pageRank(resetProbability=0.15,maxIter=10)
>>> pr.vertices.show()
+---+-------+---+------------------+
| id|   name|age|          pagerank|
+---+-------+---+------------------+
|  b|    Bob| 36|1.3772111962578364|
|  e| Esther| 32|1.1124442440992088|
|  a|  Alice| 34| 0.679533904317255|
|  f|  Fanny| 36|0.6227888037421637|
|  d|  David| 29|0.6227888037421637|
|  c|Charlie| 30|1.5852330478413725|
+---+-------+---+------------------+
```

Алгоритмы на графах: компоненты связности

- Вычисляет членство каждой вершины в компоненте связности и возвращает Vertex DataFrame, дополненный столбцом, хранящим метку компоненты связности.

```
# Все вершины должны попасть в одну компоненту связности:
>>> sc.setCheckpointDir("hdfs:///tmp")
>>> cc = g.connectedComponents()
>>> cc.show()
+---+-------+---+------------+
| id|   name|age|   component|
+---+-------+---+------------+
|  a|  Alice| 34|412316860416|
|  b|    Bob| 36|412316860416|
|  c|Charlie| 30|412316860416|
|  d|  David| 29|412316860416|
|  e| Esther| 32|412316860416|
|  f|  Fanny| 36|412316860416|
+---+-------+---+------------+
```

Прочие алгоритмы на графах

- Shortest paths: нахождение кратчайшего пути от каждой вершины до заданного списка вершин-ориентиров.
- Strongly Connected components: вычисление компонент сильной связности.
- Triangle count: подсчет количества треугольников, в состав которых входит каждая вершина.
  - Подсчет треугольников обычно используется в качестве обнаружения и подсчета сообществ в графе социальной сети.
  - Треугольник - это набор из трех вершин, где каждая вершина имеет отношение к двум другим вершинам.
- Label Propagation Algorithm (LPA): обнаружение сообществ в графе

Документация

- SQL Programming Guide: https://spark.apache.org/docs/latest/sql-programming-guide.html
- GraphFrames: https://graphframes.github.io/graphframes/docs/_site/index.html