An introduction to Spark GraphFrame with examples analyzing the Wikipedia link graph

The Spark GraphFrame is an incredibly powerful tool for performing distributed computations with large graphical data. This article introduces the GraphFrame abstraction and shows how it can be leveraged to analyze the graph formed by the links between Wikipedia articles.

Matt Hagy | Follow
Feb 22 · 6 min read

The Spark GraphFrame is a powerful abstraction for processing large graphs using distributed computing. It provides a plethora of common graph algorithms including label propagation and PageRank. Further, it provides the foundations for implementing complex graph algorithms, including a robust implementation of the Pregel paradigm for graph processing. Anyone who's interested in working with large graphs should learn how to apply this powerful tool.

In this article, I'll introduce the basics of GraphFrame and demonstrate how to use this tool through several examples. These examples consider the link graph between Wikipedia articles and I demonstrate how to analyze this graph by leveraging the GraphFrame abstraction.

Note that the examples in this post build off some more elementary Spark concepts such a DataFrames. Additionally, we use basic Scala code to demonstrate algorithms. Only small and simple examples are shown so that one doesn't need to be well familiar with these concepts to learn about the power of the GraphFrame abstraction.

Let's dive right in and consider how to create a GraphFrame. I'll start by introducing the Wikipedia link data in a basic Spark RDD. Each element of this RDD is a single Wikipedia article page represented with the following Scala class.

```
case class Page(title: String, links: Seq[String])
```

page.scala hosted with ♥ by GitHub — view raw

Note that each link is the title of another page. I.e., each page knows all the other pages that it links to by title.

We begin with a single RDD `pages` that contains 19,093,693 Wikipedia article pages. From that, we generate two Spark DataFrames, one containing of the vertices (i.e., page nodes) and the other consisting of the directed edges.

```
val vertices = pages.map(_.title).toDF("id")

case class Edge(src: String, dst: String)
val edges = {
  pages
  .flatMap(p => p.links.map(l => Edge(p.title, l)))
  .toDF()
}
```

gf_vertices_and_edges.scala hosted with ♥ by GitHub — view raw

Note that there are 206,181,091 directed edges in this graph.

Next, we create a GraphFrame using these two DataFrames.

```
val gf = GraphFrame(vertices, edges)
```

graphframe.scala hosted with ♥ by GitHub — view raw

And that's all we have to do to access this powerful abstraction. We can now start using some of the built in graph algorithms to analyze the Wikipedia link graph.

Let start by computing something simple: the Wikipedia pages with the largest number of outbound links. To this end, we can use the GraphFrame method `outDegrees`, which is a computed DataFrame that corresponds to the number of outbound edges for each vertex. Since it's a DataFrame, we can use the `orderBy` method and `limit` to select the top 10.

```
gf.outDegrees.orderBy("outDegree".desc).limit(10)
```

gf_out_degrees.scala hosted with ♥ by GitHub — view raw

This gives the following results.

Search this file...

id	outDegree
Wikipedia:WikiProject Medicine/Lists of pages/Articles	34140
Wikipedia:WikiProject Medicine/Popular pages for 2015b	29071
Draft:List of the prefectures life of Germany	24747
Draft:List of the prefectures life of France	24522
Wikipedia:ILS/Secondary/sf	24101
Wikipedia:ILS/First half	23358
Wikipedia:WikiProject Monuments/List of mountains	22767
Wikipedia:WikiProject Articles/Broken section anchors	21135
Wikipedia:WikiProject Cycling/Cycling road links	18769
Wikipedia:WikiProject Anatomy/Lists of pages/Articles	18687

out_degrees.csv hosted with ♥ by GitHub — view raw

Interestingly, we can see that many special "Wikipedia:" prefix pages have the highest number of outbound links.

Next, let's consider the number of inbound edges for each page and find the top 10 linked-to pages using the corresponding GraphFrame method, `inDegrees`.

```
gf.inDegrees.orderBy("inDegree".desc).limit(10)
```

in_degrees.scala hosted with ♥ by GitHub — view raw

Search this file...

id	inDegree
United States	343728
France	157161
Animal	146740
India	138720
World War II	134955
Germany	129348
United Kingdom	118276
Latin language	116076
Arthropod	112928
England	110503

in_degrees.csv hosted with ♥ by GitHub — view raw

We can see that locations, including countries and cities, are among the most heavily linked to pages.

Now let's explore some more complex graph computations. Let's consider the most heavily-linked to article, "United States", and find other articles in the link graph that are further away from this article in terms of the number of links you have to follow to arrive at "United States".

To this end, we can use the GraphFrame method, `shortestPaths`, which takes a collection of landmark vertices and returns the path length for every vertex in the graph to every landmark vertex.

```
val us_distance = gf.shortestPaths.landmarks(List("United States").run()

// Exclude articles that have a colon in their titles
val stringContainsColon = udf((s: String) => s.contains(":"))

// Impute the map of landmark -> path length
val getUSDistance = udf((v: Map[String, Int]) => v.getOrElse("United States", -1))

(us_distance
  .filter(!stringContainsColon("id"))
  .withColumn("us_distance", getUSDistance("distances"))
  .withColumn("us_distance" - 0)
  .orderBy("us_distance".desc)
  .limit(10))
```

wiki_us_distance.scala hosted with ♥ by GitHub — view raw

Search this file...

id	us_distance
Lumbosacral arches	10
Anous tenuirostris	10
Lumbosacral arch (disambiguation)	10
Dismal scapular vessels	9
Lachrimal fossa (disambiguation)	9
Lumbosacral arch	9
Dismal scapular (disambiguation)	9
Successive pairs	9
Lune chosser	9
The Art of Kissing Properly (album)	9

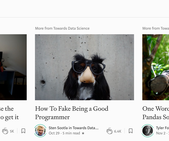
us_distance.csv hosted with ♥ by GitHub — view raw

It is interesting that there are only three articles that are a full 10 links removed from the "United States" article in terms of shortest path length.

Let's go one step further and compute the number of pages at each path length. I.e., how many Wikipedia articles are 1 link removed from "United States", how many are 2 links removed, etc.

```
display(
  us_distance
  .filter(!stringContainsColon("id"))
  .withColumn("us_distance", getUSDistance("distances"))
  .groupBy("us_distance")
  .count()
  .orderBy("us_distance")
)
```

us_dist_distance.scala hosted with ♥ by GitHub — view raw



(Note, that I'm using the built-in `display` function of the Databricks notebook. I'll talk more about the joy of using Databricks in a future post.)

Here we can see that there are millions of pages that are between 2 and 4 links removed from the "United States" article. There are also a vanishingly small number of articles that are six or more links removed. Further, the -1 path length denotes articles that aren't at all connected to the "United States" article at all and there are roughly a million pages that meet this criterion.

Lastly, no demonstration of GraphFrame would be complete without showing how easily it can be used to perform the PageRank algorithm.

```
val gr = gf.pageRank.resetProbability(0.15).maxIter(5).run()
gr.vertices.orderBy("pagerank".desc).limit(10)
```

wiki_page_rank.scala hosted with ♥ by GitHub — view raw

The page rank of the top 10 articles is:

Search this file...

id	pagerank
United States	31358.4
World War II	14776.8
France	13613.1
United Kingdom	10283.8
The New York Times	12575.4
Germany	12130.1
Race and ethnicity in the United States Census	11639.6
List of sovereign states	10860.8
New York City	10413.3
India	10260.6

wiki_page_rank.csv hosted with ♥ by GitHub — view raw

It may not be surprising that "United States" has the highest vertex rank as we know it's the most heavily linked-to article.

• • •

I hope these examples have helped to convince you that GraphFrame is a powerful abstraction for performing distributed computation on large graphs. There are additional more advanced concepts that I hope to share in future articles, including how to implement custom graph algorithms using the Pregel compute paradigm with GraphFrame.

Update: If you've found the Scala examples in this article interesting and would like to learn more about this powerful programming language you can check out my article, Quickly learning the basics of Scala through Structure and Interpretation of Computer Programs examples (Part 1).

Spark | GraphFrame | Data Science | Analytics | Graph

WRITTEN BY
Matt Hagy
Software Engineer and Fmr. Data Scientist and Manager, Ph.D. in Computational Statistical Chemistry (matthagy.com)
Follow