به‌نام خداوند بخشنده و مهربان

تمرین اول: مقدمه‌ای بر اسپارک
درس: پایگاه‌داده پیشرفته

استاد: محمدعلی نعمت‌بخش
دستیاران: فاطمه ابراهیمی، پریسا لطیفی، امیر سرتیپی

نام و نام‌خانوادگی: نگین شمس
آدرس گیت: https://github.com/NeginShams/Spark.git

- **لطفا پاسخ تمارین حتما در سامانه‌ی کوئرا ارسال شود.**
- **لطفا پاسخ‌های خود را در خود سند سوال نوشته و در قالب یک فایل PDF ارسال کنید.**
- **نام سند ارسالی HW-{homework number}-{Name Family}-{student number}**
- **تمامی فایل‌های مورد نیاز این تمرین در این لینک قابل دسترس است.**
- **خروجی از هر مرحله‌ی تمرین را در سند خود بارگذاری کنید.**
- **کد + سند را در گیت بارگذاری کرده و لینک آن را در سند قرار دهید.**

در این تمرین، هدف ما آشنایی Action و Transformation در موتور تحلیلی Spark است.

۱. منظور از Lazy Evaluation در Spark چیست؟ این مفهوم را همراه با یک مثال توضیح دهید.

در عملیات transformation به‌صورت lazy انجام می‌شود. یعنی برخی عملیات‌ها بلافاصله اجرا نشده و اجرای آن‌ها تا زمانی‌که نیاز نباشد به‌تعویق می‌افتد. به‌عبارت دیگر تا وقتی که نیاز به اجرای یک action بر روی داده‌ها نباشد، اجرا آغاز نخواهد شد. Spark با استفاده از گراف DAG (Directed Acyclic Graph) عملیات‌های مختلف وارد شده توسط کاربر را نگهداری می‌کند. گره‌های این گراف داده‌های RDD و یال‌های آن نشان‌دهنده عملیاتی است که بر روی داده‌ها اجرا خواهند شد. زمانی‌که یک action فراخوانی می‌شود، گراف DAG ایجاد شده به یک زمان‌بند داده می‌شود که آن را به چندین فاز اجرایی تقسیم می‌کند. این قابلیت باعث می‌شود که امکان تصمیم‌گیری در مورد بهینه‌سازی اجرا وجود داشته باشد . به‌عنوان یک مثال می‌توان به موردی اشاره کرد که یک transformation مانند map بر روی یک RDD اجرا می‌شود و عدد یک به تمامی عناصر RDD اضافه می‌شود. حاصل این transformation یک RDD جدید است اما بلافاصله اجرا نمی‌شود. سپس یک transformation دیگر بر روی RDD انجام شده و با استفاده از تابع map، ۹ واحد به تمام عناصر اضافه می‌شود. در این مثال به‌جای این‌که دو عدد مختلف دو بار به تمام عناصر اضافه شود، مجموع آن‌ها در یک گام به تمام عناصر افزوده می‌شود. به‌منظور بهینه‌سازی، افزودن یک و ۹ در دو گام متفاوت اجرا نمی‌شود بلکه تنها زمانی که نیاز به اجرای یک action مانند collect باشد، عدد ۱۰ به تمامی عناصر افزوده می‌شود. این موضوع را

می‌توان با استفاده از تابع todebugstring نشان داد. این تابع مقدار lineage را بازمی‌گرداند. Lineage یک گراف است که نشان می‌دهد هر RDD با استفاده از چه RDD دیگری ایجاد شده است [۱]. شکل زیر نشان دهنده‌ی مثال مذکور است.

```
mylist = [1,2,3,4,5,6,7,8,9,10]

first_rdd = spark.sparkContext.parallelize(mylist)

first_rdd
```

```
ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:274
```

```
In [3]: second_rdd = first_rdd.map(lambda x : x+1)

        print(second_rdd)

        print(second_rdd.toDebugString())
```

```
PythonRDD[1] at RDD at PythonRDD.scala:53
b'(4) PythonRDD[1] at RDD at PythonRDD.scala:53 []\n |  ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:274 []'
```

```
In [5]: third_rdd = second_rdd.map(lambda x : x+9)

        print(third_rdd)

        print(third_rdd.toDebugString())
```

```
PythonRDD[2] at RDD at PythonRDD.scala:53
b'(4) PythonRDD[2] at RDD at PythonRDD.scala:53 []\n |  ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:274 []'
```

```
In [9]: third_rdd.collect()
Out[9]: [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

۲. منظور از Narrow Transmitaion (NT) و Wide Transmitation (WT) را در Spark همراه با یک مثال بیان کنید. تفاوت اصلی این دو مفهوم چیست؟

هر RDD به‌طور ذاتی غیرقابل تغییر است و با استفاده از transformation می‌توان با اعمال تغییراتی بر روی آن، یک RDD جدید به‌دست آورد. در اسپارک عملیات transformation به دو دسته تقسیم می‌شود. در narrow transformation برای به‌دست آوردن یک partition از RDD خروجی، نیاز به داده‌های تنها یک بخش یا partition از RDD ورودی یا والد است. اما در Wide transformation برای به‌دست آوردن رکوردهای یک partition از خروجی، ممکن است نیاز به عناصر چندین partition از RDD والد باشد. به این پدیده shuffle نیز گفته می‌شود. به‌عنوان مثال در عملیات map یک تابع لامبدا بر روی تمام عناصر RDD اعمال می‌شود و عناصر هر بخش با استفاده از عناصر بخش متناظر در والد به‌دست می‌آیند. از طرف دیگر برای عملیاتی مانند groupby برای به‌دست آوردن عناصر یک partition خروجی نیاز به عناصر چندین partition از والد است زیرا ممکن است عناصر مربوط به هر گروه در partition های مختلف پخش شده باشند. WT معمولا از NT پرهزینه‌تر است. اسپارک برای NT از یک مکانیزم پایپ‌لاین استفاده می‌کند و می‌تواند آن را

به‌صورت مستقل در حافظه اصلی اجرا کند اما برای shuffle نیاز است که نتایج میانی در دیسک نوشته شود. هم‌چنین تعداد partition های RDD جدید ایجاد شده با NT دقیقا برابر تعداد partition های والد آن است. این موضوع سبب می‌شود محاسبه مجدد آن‌ها درصورت بروز خطا ساده‌تر باشد [۲].

۳. با توجه به سوال پیشین، ۴ مورد از NT، WT و Action هایی که در اسپارک وجود دارند نام ببرید.

نمونه‌های NT: map، filter، union، flatMap و mapPartiion

نمونه‌های WT: groupByKey، aggregate، aggregateByKey، join و repartition

نمونه‌های Action: collect، count، first، min، max و countByValue

۴. برای آشنایی بیشتر با مفاهیم بیان شده و مقدمه‌ای بر توابع عملیات‌های زیر را انجام داده و خروجی هریک به همراه بلاک کد آن را گزارش دهید. مثالی از خروجی برای هر بخش نمایش داده شده است.

- برای کار با اسپارک، کتاب‌خانه‌ای با نام pyspark وجود دارد.
- نوت‌بوکی بر روی گوگل کولب ایجاد کرده و این کتاب‌خانه را فراخوانی کنید.

  - برای استفاده از این کتاب‌خانه ابتدا با دستور pip install pyspark آن را نصب نموده و سپس با عبارت import pyspark آن را فراخوانی می‌کنیم.

- سپس یک لیست ۵۰ تایی از یک موضوع را برای خود درست کنید. برای مثال لیستی از (کتاب‌ها، نرم‌افزارها و ...)

  - لیست ۵۰ تایی مورد استفاده در شکل زیر قابل مشاهده است و مربوط به نام پنجاه اثر نقاشی معروف می‌باشد.

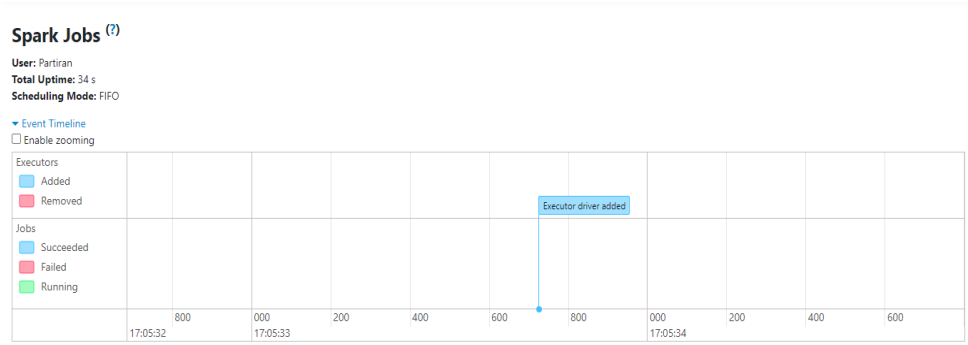
```
[ ] paintings = ['Mona Lisa', 'Girl with a Pearl Earring', 'The Last Supper', 'The Scream', 'Creation of Adam', 'The Third of May',
                 'Olympia', 'School of Athens', 'The Arnolfini Marriage', 'The Birth of Venus', 'Sistine Chapel Ceiling',
                 'Portrait of Madame Recamier', 'Massacre of the Innocents', 'Portrait of Dora Maar', 'Dogs Playing Poker', 'Primavera',
                 'The Sleeping Gypsy', 'Napoleon crossing the Alps', 'The Liberty leading the people', 'The Grand Odalisque',
                 'Les Demoiselles d'Avignon', 'American Gothic', 'Cafe Terrace at Night', 'The Son of Man', 'Bal du Moulin de la Galette',
                 'Whistler's Mother', 'Portrait de L'artiste Sans Barbe', 'The Kiss', 'The Flower Carrier', 'The Gleaners', 'The Swing',
                 'The Dance', 'The Tower of Babel', 'View of Toledo', 'The Triumph of Galatea', 'Impression, Sunrise', 'A Sunday Afternoon',
                 'Three Musicians', 'Las Meninas', 'Landscape with the Fall of Icarus', 'Water Lilies', 'No. 5, 1948', 'Luncheon on the Boating Party',
                 'The Persistence of Memory', 'Night Watch', 'Guernica', 'Beheading of Saint John', 'Starry Night', 'Royal Red and Blue', 'Composition']

[ ] len(paintings)

    50
```

- لیست خود را به RDD تبدیل کنید.

- ابتدا یک spark session ساخته می‌شود. برای تبدیل لیست به RDD از تابع parallelize() استفاده می‌شود.

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("app").config('spark.ui.port', '4050').getOrCreate()

paintings_rdd = spark.sparkContext.parallelize(paintings)
```

پس از اجرای این دستور صفحه‌ی Spark Web UI (localhost4040) به‌صورت زیر خواهد بود:



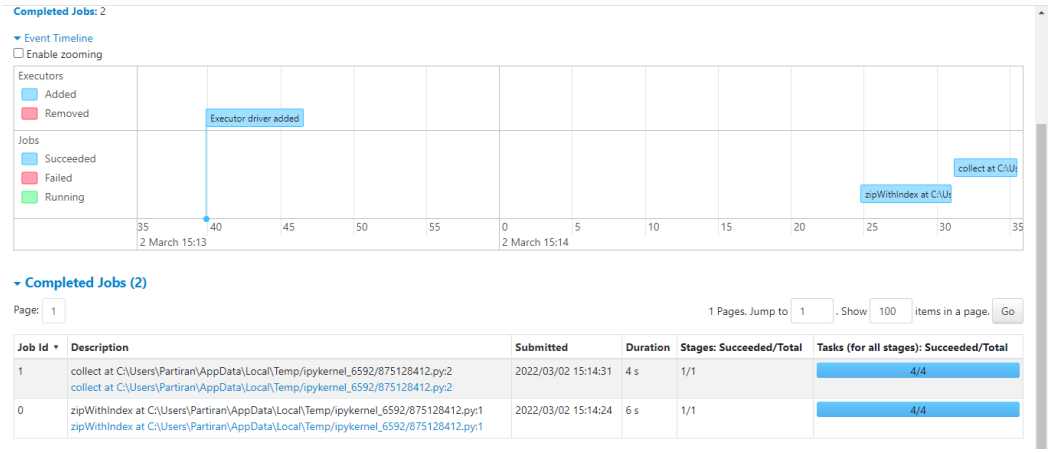
- با کمک دستور filter بر روی RDD، از آن برای بازیابی عنصر ۲۰ام لیست خود استفاده کنید. (برابر با عنصر ۲۰ام باشد)

- به‌منظور یافتن عنصر بیستم با استفاده از دستور فیلتر، ابتدا از تابع zipwithindex() استفاده نموده تا شماره اندیس عناصر به RDD اضافه شود. سپس با استفاده از دستور filter سطری از RDD که اندیس آن برابر ۱۹ باشد (عنصر بیستم) یافت می‌شود. سپس با استفاده از تابع map تنها قسمتی که شامل نام عنصر است برگردانده می‌شود. در انتها از تابع collect() برای دریافت عنصر داخل RDD نهایی استفاده شده است.

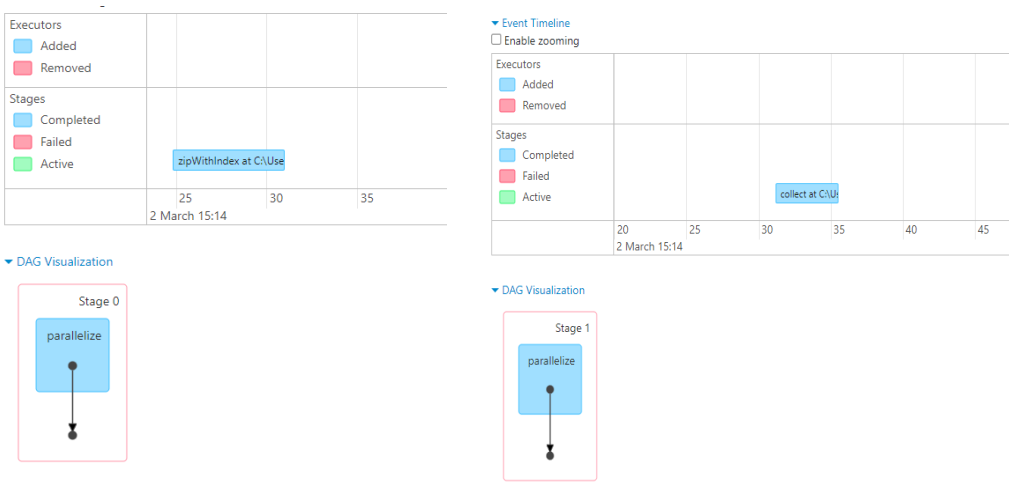
finding twentieth element

```
[5]  zip_words = paintings_rdd.zipWithIndex()
     elem = zip_words.filter(lambda x: x[1]==19).map(lambda x: x[0]).collect()
     print(elem)

     ['The Grand Odalisque']
```

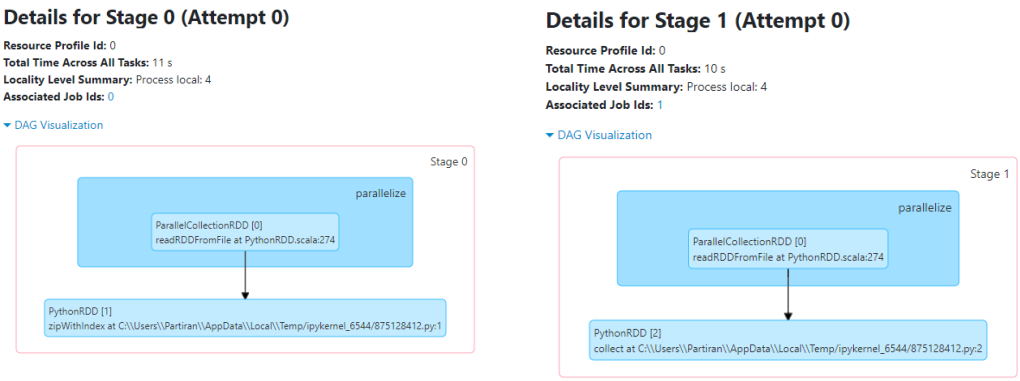
توالی انجام عملیات به‌صورت زیر می‌باشد:



این دستور شامل دو job است که هر کدام از یک stage تشکیل شده‌اند. هرگاه در یک job نیاز به shuffle باشد، یک stage جدید در آن ایجاد می‌شود. Job های مربوط به این سوال هر کدام تنها از یک stage تشکیل شده‌اند. در ابتدا zipwithindex اجرا شده و پس از اتمام اجرای آن collect اجرا می‌شود.

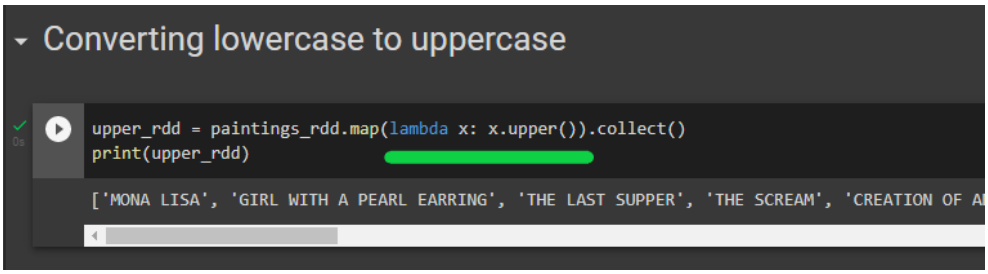


هم‌چنین DAG Visulization این دو فاز به‌صورت جزئی‌تر به‌صورت زیر می‌باشد:

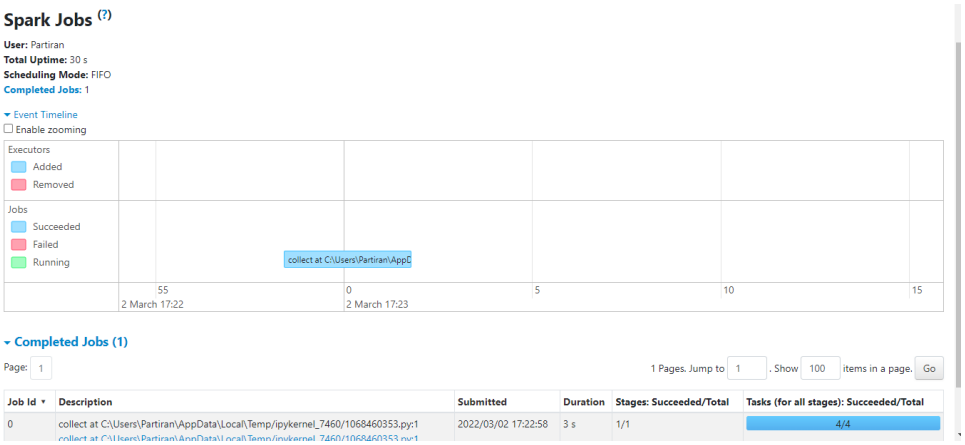


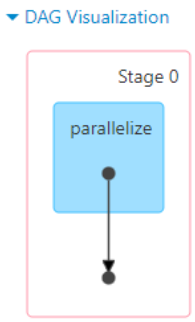
- با کمک map تمامی عناصر لیست خود را به حروف بزرگ تبدیل و آن را بازیابی کنید.

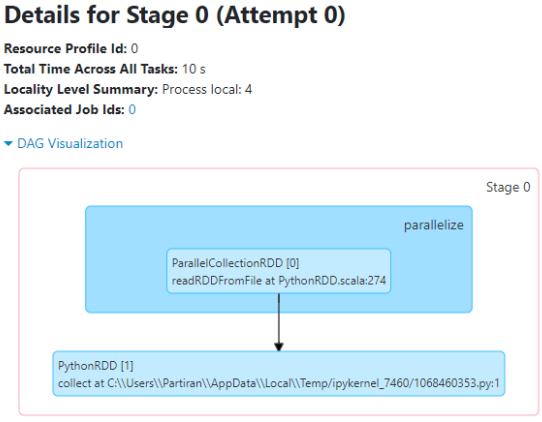
- با استفاده از map می‌توان یک تابع را بر روی تمام عناصر RDD اعمال نموده و یک RDD جدید ایجاد نمود. بنابراین با استفاده از تابع مشخص شده در شکل زیر می‌توان تمام عناصر را به حروف بزرگ تبدیل نمود.



این دستور از یک job تشکیل شده است زیرا تنها یک action یعنی collect مورد نیاز بوده است. هم‌چنین این job نیز تنها از یک stage تشکیل شده است زیرا عملیات suffle صورت نگرفته است.







- با کمک دستور groupby و map، لیست خود را بر اساس اولین کاراکتر آن دسته بندی کنید.

با استفاده از تابع groupby می‌توان کلمات را براساس حرف اول آن‌ها دسته‌بندی نمود. اگر نیازی به مرتب‌سازی نباشد، اجرای دستور زیر کافی خواهد بود.

Group by first character (without sorting)

```
[ ] result = paintings_rdd.groupBy(lambda x: x[0]).map(lambda x:(x[0],list(x[1]))).collect()
    result
```

- همچنین درصورتی‌که نیاز به مرتب نمودن نتایج باشد، با استفاده از تابع map می‌توان یک تابع لامبدا تعریف نمود که عناصر داخل هر گروه را مرتب کند. با استفاده از تابع sortBy نیز می‌توان گروه‌ها را به‌صورت مرتب‌شده نمایش داد.

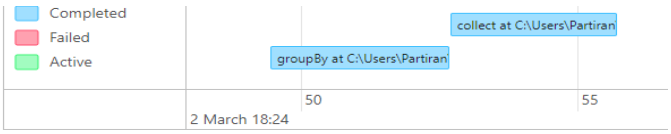
Group by first character

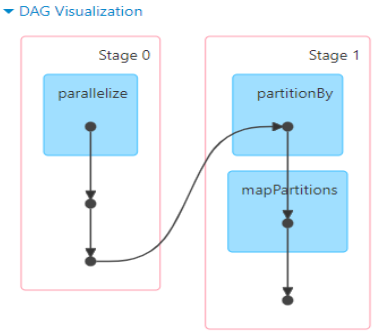
```
result = paintings_rdd.groupBy(lambda x: x[0]).map(lambda x:tuple([x[0],sorted(x[1])])).sortBy(lambda x: x[0]).collect()
result
```

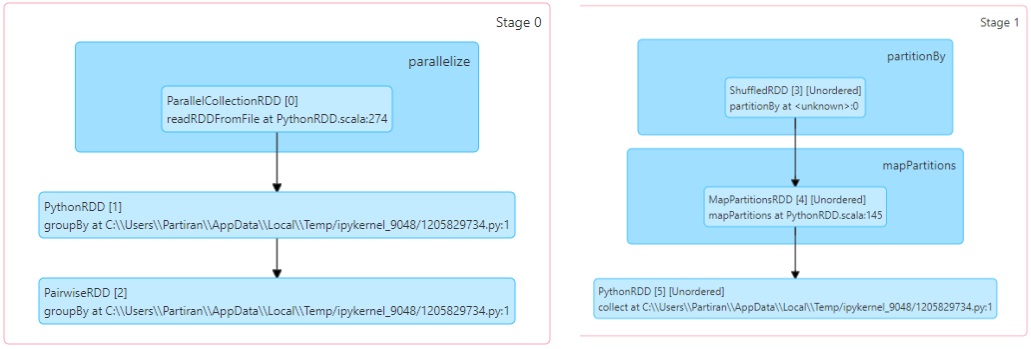
```
[('A', ['A Sunday Afternoon', 'American Gothic']),
 ('B', ['Bal du Moulin de la Galette', 'Beheading of Saint John']),
 ('C', ['Cafe Terrace at Night', 'Composition', 'Creation of Adam']),
 ('D', ['Dogs Playing Poker']),
 ('G', ['Girl with a Pearl Earring', 'Guernica']),
 ('I', ['Impression, Sunrise']),
 ('L',
  ['Landscape with the Fall of Icarus',
   'Las Meninas',
   'Les Demoiselles d'Avignon',
   'Luncheon on the Boating Party']),
 ('M', ['Massacre of the Innocents', 'Mona Lisa']),
 ('N', ['Napoleon crossing the Alps', 'Night Watch', 'No. 5, 1948']),
 ('O', ['Olympia']),
 ('P',
  ['Portrait de L'artiste Sans Barbe',
   'Portrait of Dora Maar',
   'Portrait of Madame Recamier',
   'Primavera']),
 ('R', ['Royal Red and Blue']),
 ('S', ['School of Athens', 'Sistine Chapel Ceiling', 'Starry Night']),
```

```
('T',
 ['The Arnolfini Marriage',
  'The Birth of Venus',
  'The Dance',
  'The Flower Carrier',
  'The Gleaners',
  'The Grand Odalisque',
  'The Kiss',
  'The Last Supper',
  'The Liberty leading the people',
  'The Persistence of Memory',
  'The Scream',
  'The Sleeping Gypsy',
  'The Son of Man',
  'The Swing',
  'The Third of May',
  'The Tower of Babel',
  'The Triumph of Galatea',
  'Three Musicians']),
('V', ['View of Toledo']),
('W', ['Water Lilies', 'Whistler's Mother'])]
```

انجام عملیات برای این دستور در حالت بدون مرتب‌سازی شامل یک job است زیرا تنها یک action یعنی collect وجود داشته است. این job از دو stage تشکیل شده است زیرا groupBy یک WT است و نیاز به Shuffling دارد.







- عملیات map و reduce را بر روی یک متن نسبتا بلند پس از تبدیل توکن‌های آن به rdd، انجام دهید.

در ابتدا با استفاده از sparkContext.textFile یک فایل متنی خوانده می‌شود. درصورتی‌که نیاز به مرتب‌سازی نتایج نباشد می‌توان از دستور نشان داده شده در شکل زیر استفاده نمود. در ابتدا از عملیات flatMap استفاده شده و کلمات موجود در هر خط را باز گردانده می‌شود. تفاوت flatMap و map در این است که flatMap میتواند به‌ازای هر عنصر RDD چندین عنصر جدید ایجاد نماید. سپس با استفاده از تابع Map هر کلمه به‌صورت یک جفت (word, 1) تبدیل می‌شود و در مرحله بعد با استفاده از reduceByKey می‌توان یک تابع لامبدا تعریف نمود که تعداد رخداد هر کلید را محاسبه می‌کند.

```
text_file = spark.sparkContext.textFile("/content/drive/MyDrive/van_gogh.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda x, y: x + y)
output = counts.collect()
output
```

هم‌چنین می‌توان بدون استفاده از flatMap این عملیات را انجام داد. می‌توان ابتدا متن را خوانده و کلمات آن را جدا نمود و در یک متغیر لیست قرار داد. سپس لیست را به RDD تبدیل نمود. در شکل زیر هم‌چنین علائم نگارشی مانند نقطه از متن حذف شده‌اند و نتایج نیز براساس بیشترین تعداد رخداد مرتب شده‌اند.

Map Reduce

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

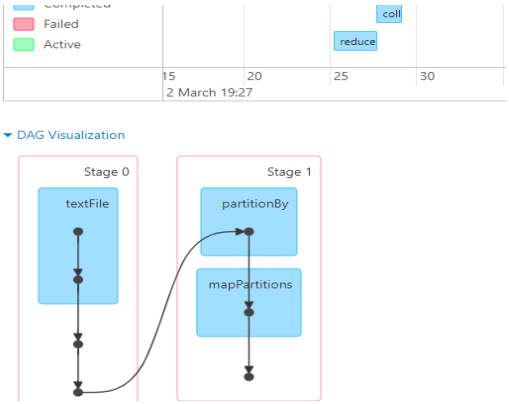
```
import string

# read the text file
file_content = open('/content/drive/MyDrive/van_gogh.txt', "r", encoding='utf-8-sig').read()

new_text = file_content.translate(str.maketrans('', '', string.punctuation))

tokens = new_text.split()

#turn into rdd
text_rdd = spark.sparkContext.parallelize(tokens)

wordCounts = text_rdd.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b).sortBy(lambda x: -x[1]).collect()
wordCounts
```

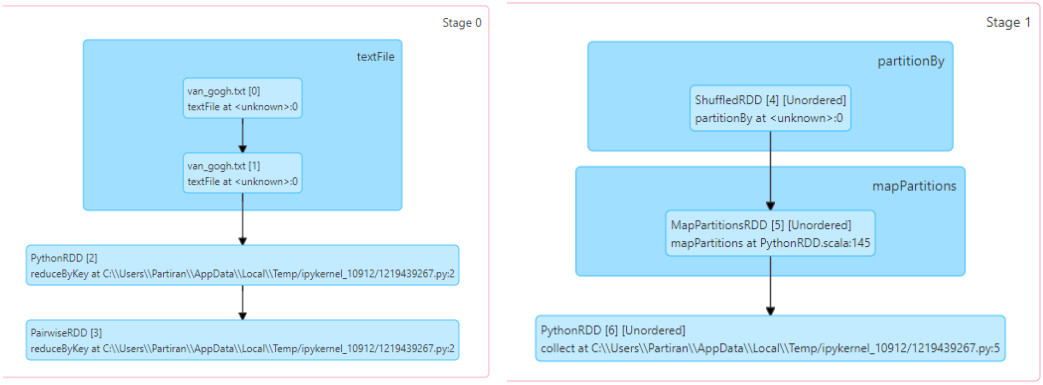
```
[('the', 643),
 ('and', 494),
 ('of', 479),
 ('a', 429),
 ('in', 403),
 ('to', 326),
 ('Van', 257),
 ('Gogh', 235),
 ('his', 235),
 ('he', 186),
 ('with', 180),
 ('was', 150),
 ('The', 118),
 ('as', 106),
 ('at', 98),
 ('that', 97),
 ('on', 95),
 ('by', 79),
 ('In', 78),
 ('from', 78),
 ('is', 72),
 ('He', 72),
 ('for', 71),
 ('Vincent', 64),
 ('Goghs', 63),
```

```
 ('not', 26),
 ('works', 26),
 ('where', 25),
 ('large', 25),
 ('time', 25),
 ('trees', 25),
 ('been', 25),
 ('wrote', 25),
 ('series', 24),
 ('when', 24),
 ('Amsterdam', 24),
 ('Art', 22),
 ('background', 22),
 ('May', 21),
 ('became', 21),
 ('one', 21),
 ('they', 21),
 ('right', 20),
 ('sky', 20),
 ('first', 20),
```

```
 ('landmarks', 2),
 ('area', 2),
 ('rooms', 2),
 ('furnished', 2),
 ('move', 2),
 ('Over', 2),
 ('Twelve', 2),
 ('idea', 2),
 ('café', 2),
 ('go', 2),
 ('sowing', 2),
 ('edge', 2),
 ('Poet', 2),
 ('8', 2),
 ('Alyscamps', 2),
 ('admired', 2),
 ('billiard', 2),
 ('Vineyard', 2),
 ('simple', 2),
 ('lit', 2),
 ('damage', 2),
```

درصورت اجرای روش اول و مشاهده نتیجه در واسط اسپارک، یک job وجود خواهد داشت که از دو فاز تشکیل شده است زیرا reduceByKey یک WT است. در ابتدا عملیات flatMap بر روی RDD اجرا شده و سپس map و پس از آن reduceByKey اجرا می‌شود. در انتها با استفاده از collect، نتایج بر روی حافظه درایور قرار می‌گیرد.





- چه تفاوتی بین Actionهای take و collect وجود دارد؟

با استفاده از collect تمام عناصر به‌صورت یک آرایه بازگردانده می‌شود. این تابع تمام عناصر را داخل حافظه driver قرار می‌دهد. اما تابع take عدد n را به‌عنوان آرگومان ورودی دریافت نموده و به تعداد n عنصر اولیه از RDD را باز می‌گرداند. برای اجرای take ابتدا نخستین partition اسکن می‌شود سپس از نتیجه به‌دست آمده برای تخمین تعداد بقیه‌ی partition هایی که باید اسکن شوند استفاده می‌شود. نتایج برگردانده شده توسط این تابع نیز در driver-memory قرار می‌گیرد [۳].

- در صورتی که بتوانید توالی انجام هریک از عملیات‌ها در اسپارک که برای هر دستور انجام می‌دهد را برای هریک از دستورات بالا نمایش دهید و باتوجه به مفاهیم سوالات قبل آن را تصویر سازی کنید، نمره اضافه‌ای دریافت خواهید کرد. (به کمک ngrok و UI Spark)

برای استفاده از قابلیت واسط گرافیکی اسپارک روی پورت ۴۰۴۰، ماژول‌های pyspark و findspark بر روی سیستم محلی نصب شده و کدها در jupyter notebook نیز اجرا شده‌اند. با استفاده از spark.sparkContext.uiWebUrl می‌توان به صفحه spark web ui دسترسی یافت. تصویر نتایج به‌دست آمده در بخش‌های قبلی قرار داده شده است.

مراجع

[1] L. Arora, "Being Lazy is Useful — Lazy Evaluation in Spark," 28 Oct 2019. [Online]. Available: https://medium.com/analytics-vidhya/being-lazy-is-useful-lazy-evaluation-in-spark-1f04072a3648.

[2] [Online]. Available: https://databricks.com/glossary/what-are-transformations.

[3] [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.take.html.