به‌نام خداوند بخشنده و مهربان

تمرین اول: مقدمه‌ای بر اسپارک
استاد: محمدعلی نعمت‌بخش
درس: پایگاه‌داده پیشرفته
دستیاران: فاطمه ابراهیمی، پریسا لطیفی، امیر سرتیپی

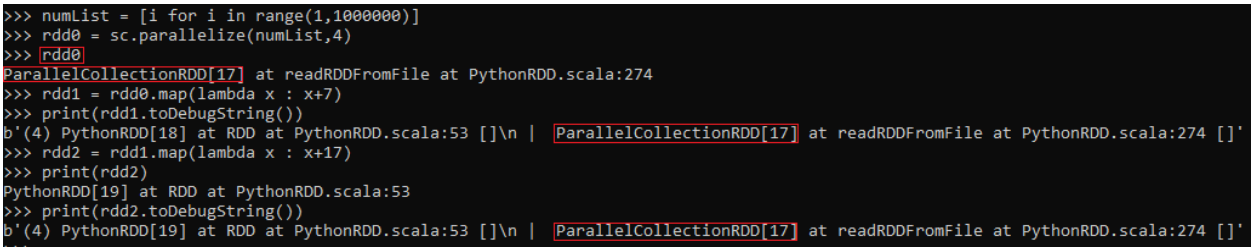
نام و نام‌خانوادگی: فاطمه مومنی
آدرس گیت: https://github.com/FatemehMomeni/BigData1.git

- **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.

منظور از Lazy Evaluation در Spark، عدم اجرای فرایند تا زمان فراخوانی یک action است. یعنی اجرای برنامه، به‌ترتیب کد نوشته شده نیست بلکه Spark تنها زمانی که لازم باشد (با فراخوانی یک action)، transformationهای مربوطه را اجرا می‌کند. با این شیوه، ممکن است بعضی transformationها ادغام شده و transformationهای غیرضروری اجرا نشوند. شکل ۱ مثالی از این مفهوم را نشان می‌دهد. پس از ایجاد لیست numList که با اعداد یک تا یک میلیون مقداردهی شده است و تبدیل آن به RDD(با تابع parallelize())، هر یک از عناصر این لیست را یک‌بار با عدد هفت و بار دیگر با عدد ۱۷ جمع می‌کنیم. با فراخوانی تابع toDebugString() که RDD Lineage (گرافی از همه RDDهای پدر این RDD) را نشان می‌دهد، مشخص می‌شود که Spark به‌جای انجام هر یک از عملیات‌های جمع به‌صورت مجزا، دو عملیات را ادغام کرده و هر یک از عناصر لیست را با عدد ۲۴ جمع می‌کند. البته برای انجام این جمع روی عناصر لیست، باید یک action فراخوانده شود. مزیت این روش، عدم انجام محاسبات غیر ضروری و در نتیجه کاهش مصرف حافظه و افزایش سرعت می‌باشد.



شکل ۱: مثال Lazy Evaluation

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؟

در Narrow Transformation تمام عناصر لازم برای ایجاد و محاسبه رکوردهای یک partition، در یک partition از RDD پدر قرار دارد و زیرمجموعه محدودی از partitionها برای محا سبه نتیجه نیاز ا ست اما در Wide Transformation، تمام عنا صر لازم برای ایجاد و محا سبه رکوردهای یک partition، ممکن ا ست در چند partition از RDD پدر قرار دا شته و partition حا صل می‌تواند در چندین partition از RDD پدر قرار دا شته با شد. برای نمونه ()filter که یک NT ا ست، آرگومان ورودی خود را که تابعی برای انجام عملیات فیلتر است، روی یک partition از RDD فراخوانی شده اعمال می‌کند اما groupByKey که یک WT است، آرگومان ورودی را روی تمام RDD که در partitionهای مختلف قرار دارد، اعمال می‌کند.

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

NT: map، flatMap، filter، sample

WT: reducedByKey، groupByKey، distinct، intersection

Action: count، collect، take، top

۴. برای آ شنایی 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.
- سپس یک لی ست ۵۰ تایی از یک مو ضوع را برای خود در ست کنید. برای مثال لی ستی از (کتاب‌ها، نرم‌افزارها و ...)
- لیست خود را به RDD تبدیل کنید.

شکل ۲ لیستی از یک موضوع و تبدیل آن به RDD و شکل ۳ خروجی عملیات collect را نشان می‌دهند.

```
[66] animations = 'Soul,Inside Out,Zootopia,Up,Toy Story4,Encanto,The Lion King,Rango,Toy Story3,The Mithcells vs the Machines,Toy Story2,Toy Story1,Cars3,Cars2,Cars1

[67] animations = animations.split(',')

[68] from pyspark import SparkContext
     sc =SparkContext.getOrCreate()

     animations_rdd = sc.parallelize(animations)
```

شکل ۲: لیست موضوع و تبدیل آن به RDD

```
animations_rdd.collect()

['Soul',
 'Inside Out',
 'Zootopia',
 'Up',
 'Toy Story4',
 'Encanto',
 'The Lion King',
 'Rango',
 'Toy Story3',
 'The Mithcells vs the Machines',
 'Toy Story2',
 'Toy Story1',
 'Cars3',
 'Cars2',
 'Cars1',
 'Ralph Breaks the Internet',
 'Captain Underpants',
 'The Croods',
 'Ice Age5',
 'Ice Age4',
 'Ice Age3',
 'Ice Age2',
 'Ice Age1',
 'Rons Gone Wrong',
 'Despicable Me2',
 'Despicable Me1',
 'Hotel Transylvania4',
 'Hotel Transylvania3',
 'Hotel Transylvania2',
 'Hotel Transylvania1',
 'WALL-E',
 'Wish Dragon',
 'Winnie the Pooh',
 'Pinocchio',
 'Finding Nemo',
 '101 Dalmatians',
 'The Jungle Book',
 'Brother Bear',
 'Dumbo',
 'Angry Birds2',
 'Big Hero',
 'Snow White and the Seventh Dwarfs',
 'Shrek',
 'Frozen',
 'Luca',
 'Turning Red',
 'Coco',
 'Rumble',
 'Hercules',
 'Moana']
```

شکل ۳: نمایش لیست

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

```
[69] anim20th_rdd = animations_rdd.filter(lambda x:x==animations[20])
```

شکل ۴: بازیابی عنصر بیستم لیست

```
[50] anim20th_rdd.collect()
```

```
['Ice Age3']
```

شکل ۵: عنصر بیستم لیست

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

```
[51] cap_anims_rdd = animations_rdd.map(lambda x:x.upper())
```

شکل ۶: تبدیل عناصر لیست به حروف بزرگ

```
[52] cap_anims_rdd.collect()
```

```
['SOUL',
 'INSIDE OUT',
 'ZOOTOPIA',
 'UP',
 'TOY STORY4',
 'ENCANTO',
 'THE LION KING',
 'RANGO',
 'TOY STORY3',
 'THE MITHCELLS VS THE MACHINES',
 'TOY STORY2',
 'TOY STORY1',
 'CARS3',
 'CARS2',
 'CARS1',
 'RALPH BREAKS THE INTERNET',
 'CAPTAIN UNDERPANTS',
 'THE CROODS',
 'ICE AGE5',
 'ICE AGE4',
 'ICE AGE3',
 'ICE AGE2',
 'ICE AGE1',
 'RONS GONE WRONG',
 'DESPICABLE ME2',
 'DESPICABLE ME1',
 'HOTEL TRANSYLVANIA4',
 'HOTEL TRANSYLVANIA3',
 'HOTEL TRANSYLVANIA2',
 'HOTEL TRANSYLVANIA1',
 'WALL-E',
 'WISH DRAGON',
 'WINNIE THE POOH',
 'PINOCCHIO',
 'FINDING NEMO',
 '101 DALMATIANS',
 'THE JUNGLE BOOK',
 'BROTHER BEAR',
 'DUMBO',
 'ANGRY BIRDS2',
 'BIG HERO',
 'SNOW WHITE AND THE SEVENTH DWARFS',
 'SHREK',
 'FROZEN',
 'LUCA',
 'TURNING RED',
 'COCO',
 'RUMBLE',
 'HERCULES',
 'MOANA']
```

شکل ۷: لیست با حروف بزرگ

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

```
[53] groupMap_anims_rdd = animations_rdd.groupBy(lambda x:x[0]).map(lambda x:(x[0],list(x[1])))
```

شکل ۸: دسته‌بندی لیست براساس اولین کاراکتر

```
[54] groupMap_anims_rdd.collect()

[('S', ['Soul', 'Snow White and the Seventh Dwarfs', 'Shrek']),
 ('R', ['Rango', 'Ralph Breaks the Internet', 'Rons Gone Wrong', 'Rumble']),
 ('C', ['Cars3', 'Cars2', 'Cars1', 'Captain Underpants', 'Coco']),
 ('W', ['WALL-E', 'Wish Dragon', 'Winnie the Pooh']),
 ('1', ['101 Dalmatians']),
 ('L', ['Luca']),
 ('I',
  ['Inside Out', 'Ice Age5', 'Ice Age4', 'Ice Age3', 'Ice Age2', 'Ice Age1']),
 ('Z', ['Zootopia']),
 ('U', ['Up']),
 ('T',
  ['Toy Story4',
   'The Lion King',
   'Toy Story3',
   'The Mithcells vs the Machines',
   'Toy Story2',
   'Toy Story1',
   'The Croods',
   'The Jungle Book',
   'Turning Red']),
 ('E', ['Encanto']),
 ('D', ['Despicable Me2', 'Despicable Me1', 'Dumbo']),
 ('H',
  ['Hotel Transylvania4',
   'Hotel Transylvania3',
   'Hotel Transylvania2',
   'Hotel Transylvania1',
   'Hercules']),
 ('P', ['Pinocchio']),
 ('F', ['Finding Nemo', 'Frozen']),
 ('B', ['Brother Bear', 'Big Hero']),
 ('A', ['Angry Birds2']),
 ('M', ['Moana'])]
```

شکل ۹: لیست دسته‌بندی شده

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

```
[72] article = Using_Data_Mining_to_Model_Player_Experience.split()
     article_rdd = sc.parallelize(article)
```

```
[75] mapReduce_rdd = article_rdd.map(lambda x:(x,1)).reduceByKey(lambda x,y:x+y)
```

شکل ۱۰: تبدیل متن به RDD و انجام عملیات نگاشت-کاهش

```
[76] mapReduce_rdd.collect()
```

```
[('ABSTRACT', 1),
 ('mining', 5),
 ('designers', 9),
 ('measure', 7),
 ('in', 43),
 ('providing', 4),
 ('quantitative', 5),
 ('These', 2),
 ('used', 15),
 ('telemetry-supported', 2),
 ('where', 6),
 ('human', 2),
 ('designer', 4),
 ('support', 1),
 ('of', 143),
 ('data.', 3),
 ('model', 11),
 ('retention', 18),
 ('as', 32),
 ('technique', 4),
 ('identifying', 1),
 ('have', 5),
```

شکل ۱۱: بخشی از خروجی عملیات نگاشت-کاهش

- چه تفاوتی بین Actionهای take و collect وجود دارد؟

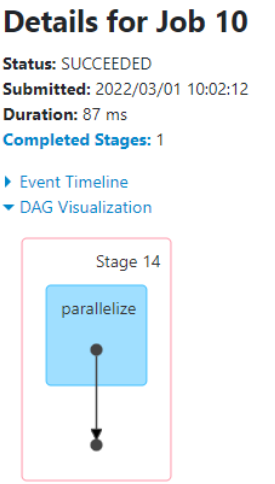
collect تمام محتوا و عناصر RDD را از همه گره‌ها به گره driver بازمی‌گرداند و RDD را به into-memory list تبدیل می‌کند. معمولا collect روی مجموعه داده‌های کوچک استفاده می‌شود؛ زیرا بازیابی آن در مجموعه داده‌های بزرگ باعث کمبود حافظه می‌شود. take تعدادی از عناصر RDD را برمی‌گرداند و سعی می‌کند تعداد partitionهایی که به آن دسترسی دارد را کاهش دهد. take خروجی را در یک سطر نمایش می‌دهد.

- در صورتی که بتوانید توالی انجام هریک از عملیات‌ها در ا سپارک که برای هر د ستور انجام می‌دهد را برای هریک از د ستورات بالا نمایش دهید و باتوجه به مفاهیم سوالات قبل آن را ت صویر سازی کنید، نمره اضافه‌ای دریافت خواهید کرد. (به کمک ngrok و UI Spark)

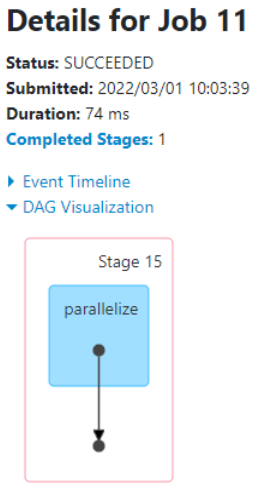
شکل ۱۲ تا شکل ۱۶ DAG هر یک از عملیات‌های بالا را نشان می‌دهند. با فراخوانی یک action، DAG ایجاد شده به DAG Scheduler که گراف را به stageهایی از وظایف تقسیم می‌کند، داده می‌شود. stageها براساس transformationها ساخته می‌شوند و NTها باهم در یک stage گروه‌بندی می‌شوند. شکل ۱۲ مربوط به ایجاد RDD است و دستور parallelize را دارد. شکل ۱۳ و

شـــکل ۱۴ به‌ترتیب عملیات‌های filter و map را روی RDD انجام می‌دهند و دارای یک stage هستند. شکل ۱۵ دستور groupBy که WT است و دستور map که NT است را دارد. پس دارای دو stage است. شکل ۱۶ مشابه شکل قبل است و دستورات map که NT است و reduceByKey که WT اســـت را دارد. در تمامی شـــکل‌ها تا زمان فراخوانی collect action، transformationها اجرا نشده و پس از فراخوانی به‌صورت بهینه اجرا می‌شوند.

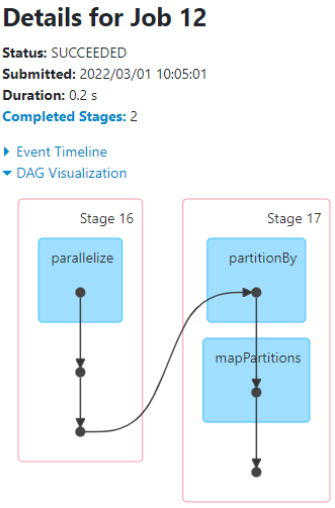
شکل ۱۲: DAG ساخت RDD



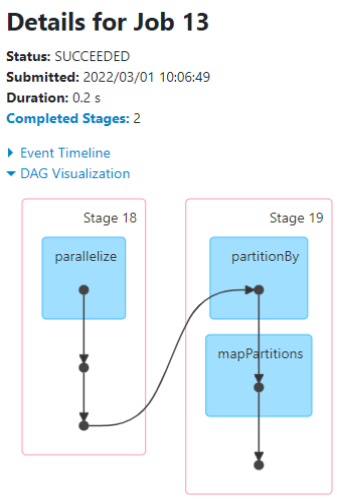
شکل ۱۳: DAG بازیابی عنصر بیستم



شکل ۱۴: DAG تبدیل عناصر لیست به حروف بزرگ



شکل ۱۵: DAG دسته‌بندی لیست براساس اولین کاراکتر



شکل ۱۶: DAG انجام عملیات نگاشت-کاهش روی متن