第四章　Spark习题解答

毕倪飞、杨溢

華東師範大學

大纲

- TopK的平方和
- 广播K均值聚类
- 基站统计
- 计算传递闭包
- 网页排名前三名
- 移动平均

题目描述

- 要求：找出出现次数最多的5个数，求这5个数出现次数的平方和
- 输入：输入数据由多行组成，每行有多个整数，并且整数之间用空格进行分隔
- 输出：出现次数最多的5个数的出现次数的平方和

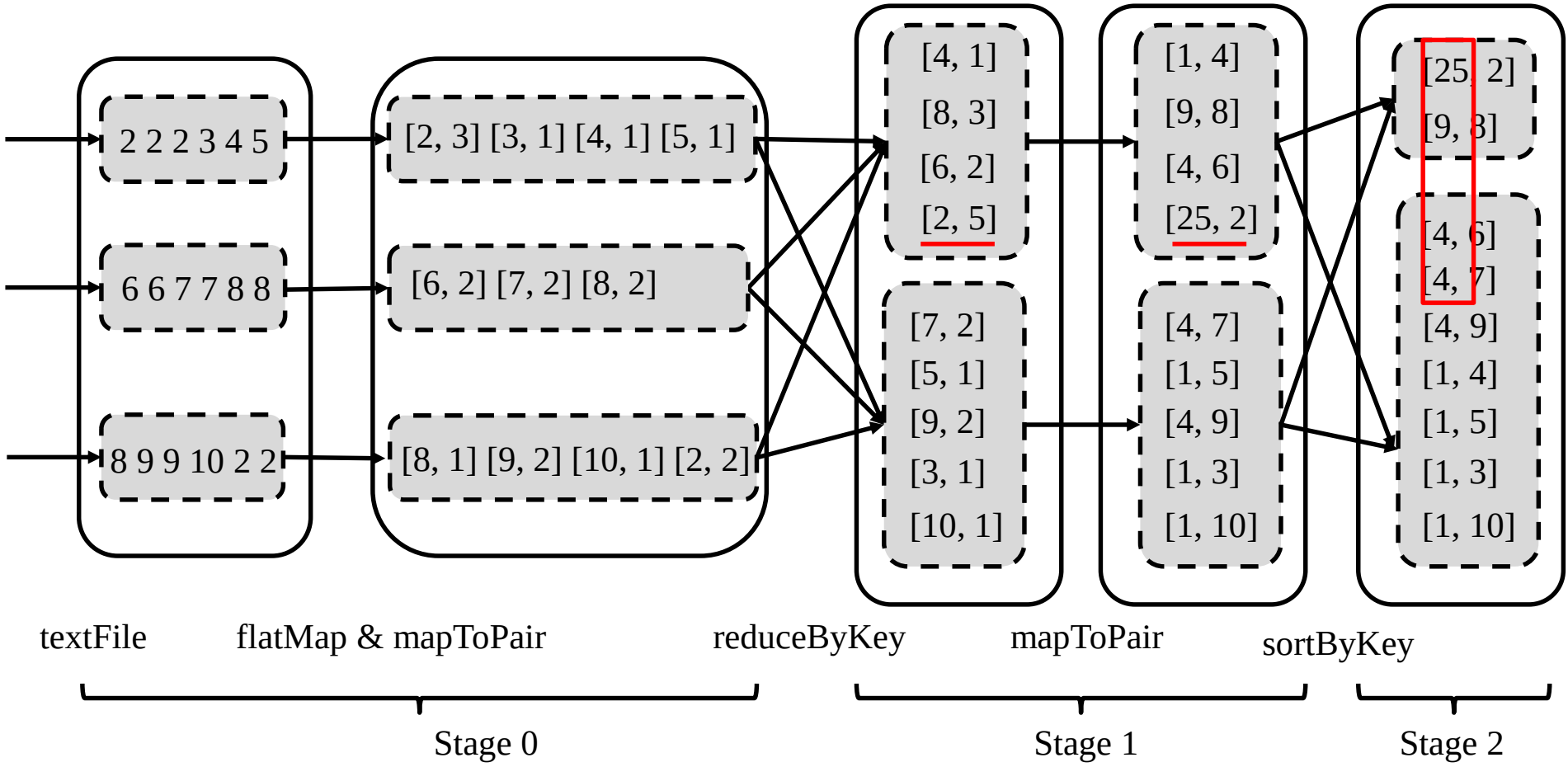
输入	输出
2 2 2 3 4 5 6 6 7 7 8 8 8 9 9 10 2 2	46

注：$5^2 + 3^2 + 2^2 + 2^2 + 2^2 = 46$

解决方案

- 统计数字出现次数
  - flatMap，按空格分割数字
  - mapToPair，将分割后的数字映射为[数字，1]
  - reduceByKey，得到[数字，出现次数]
- 交换键值并求解
  - mapToPair，交换键值并求键的平方，得到[出现次数^2，数字]
  - sortByKey，按照出现次数进行降序排序
- 取出答案
  - take，取出前5个键值对，并对键求和得到答案

TopK的平方和 运行过程



编写TopKPower方法


```
public class TopKPowerImpl extends TopKPower {

  @Override
  public int topKPower(JavaRDD<String> lines) {
    // 计算每个宝箱的发光次数，得到箱子ID<箱子ID，发光次数>
    JavaPairRDD<String, Integer> words = lines
        .flatMap(line -> Arrays.stream(line.split(" ")).iterator())
        .mapToPair(word -> new Tuple2<>(word, 1))
        .reduceByKey(Integer::sum);
    // 交换 KV 对为 <发光次数*发光次数，箱子ID>，根据发光次数的平方进行降序排序
    JavaPairRDD<Integer, String> countPower = words.mapToPair(
        tuple2 -> new Tuple2<Integer, String>(tuple2._2() * tuple2._2(), tuple2._1()))
        .sortByKey(false);

    // 取出发光次数最多的5个箱子
    List<Tuple2<Integer, String>> top5 = countPower.take(5);

    // 计算平方和
    Integer ans = 0;
    for (Tuple2<Integer, String> tuple2 : top5) {
      ans += tuple2._1();
    }

    return ans;
  }
}
```

大纲

- TopK的平方和
- 广播K均值聚类
- 基站统计
- 计算传递闭包
- 网页排名前三名
- 移动平均

广播K均值聚类

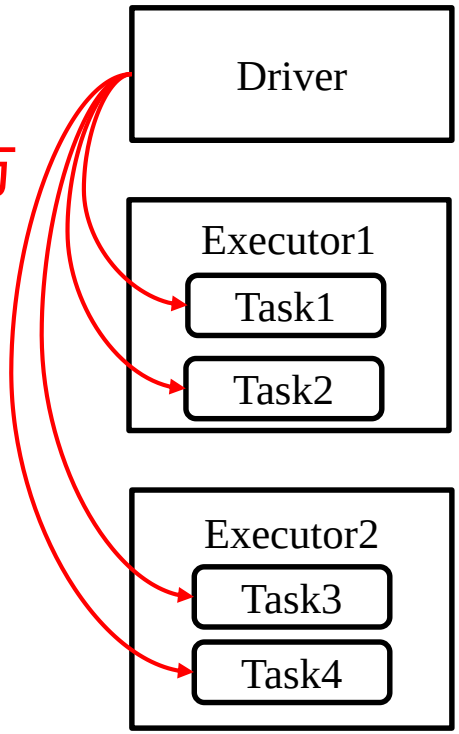
- 要求：使用广播机制实现KMeans算法。
- 输入：两个文本文件，分别保存数据集和初始聚类中心集。每行按逗号分割成两个整数，表示一个二维点。
- 输出：数据点及其所属类别

数据集	聚类中心集	聚类结果
0,0 1,2 3,1 8,8 9,10 10,7	1,2 3,1	0,0 1 1,2 1 10,7 2 3,1 1 8,8 2 9,10 2

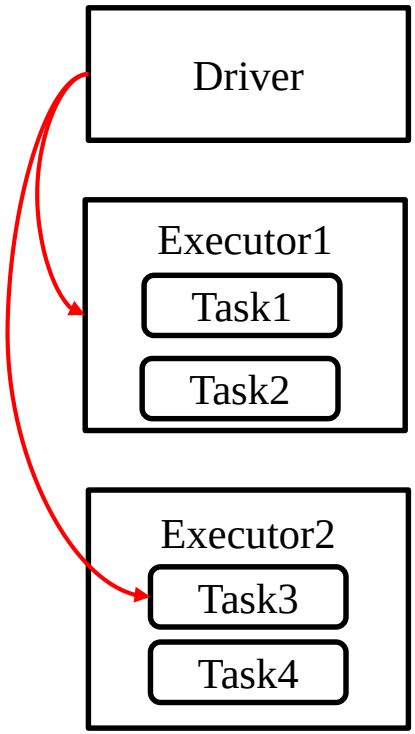
为什么用broadcast机制

- 若Executor数量小于Task数量，broadcast方式比“广播”方式传输的数据量更小

使用“广播”方式，聚类中心点会向每个Task发送一份



使用broadcast机制，聚类中心点会向每个Executor发送一份



实现抽象方法


```
@Override
public Broadcast<List<List<Double>>> createBroadcastVariable(JavaSparkContext sc,
    List<List<Double>> localVariable) {
  return sc.broadcast(localVariable);
}

@Override
public Integer closestPoint(List<Integer> p, Broadcast<List<List<Double>>> kPoints) {
  int bestIndex = 0;
  double closest = Double.MAX_VALUE;
  for (int i = 0; i < kPoints.value().size(); i++) {
    double dist = distanceSquared(p, kPoints.value().get(i));
    if (dist < closest) {
      closest = dist;
      bestIndex = i;
    }
  }
  return bestIndex;
}
```

使用sc.broadcast()方法创建广播变量

使用value()方法获取广播变量的值

大纲

- TopK的平方和
- 广播K均值聚类
- 基站统计
- 计算传递闭包
- 网页排名前三名
- 移动平均

题目描述

- 要求：计算每个人往（返）途中在每个基站停留的总时间（秒）
- 输入：每行文本包含姓名、基站名、进入基站的时刻或从基站出来的时刻
- 输出：姓名、基站名、总停留时间

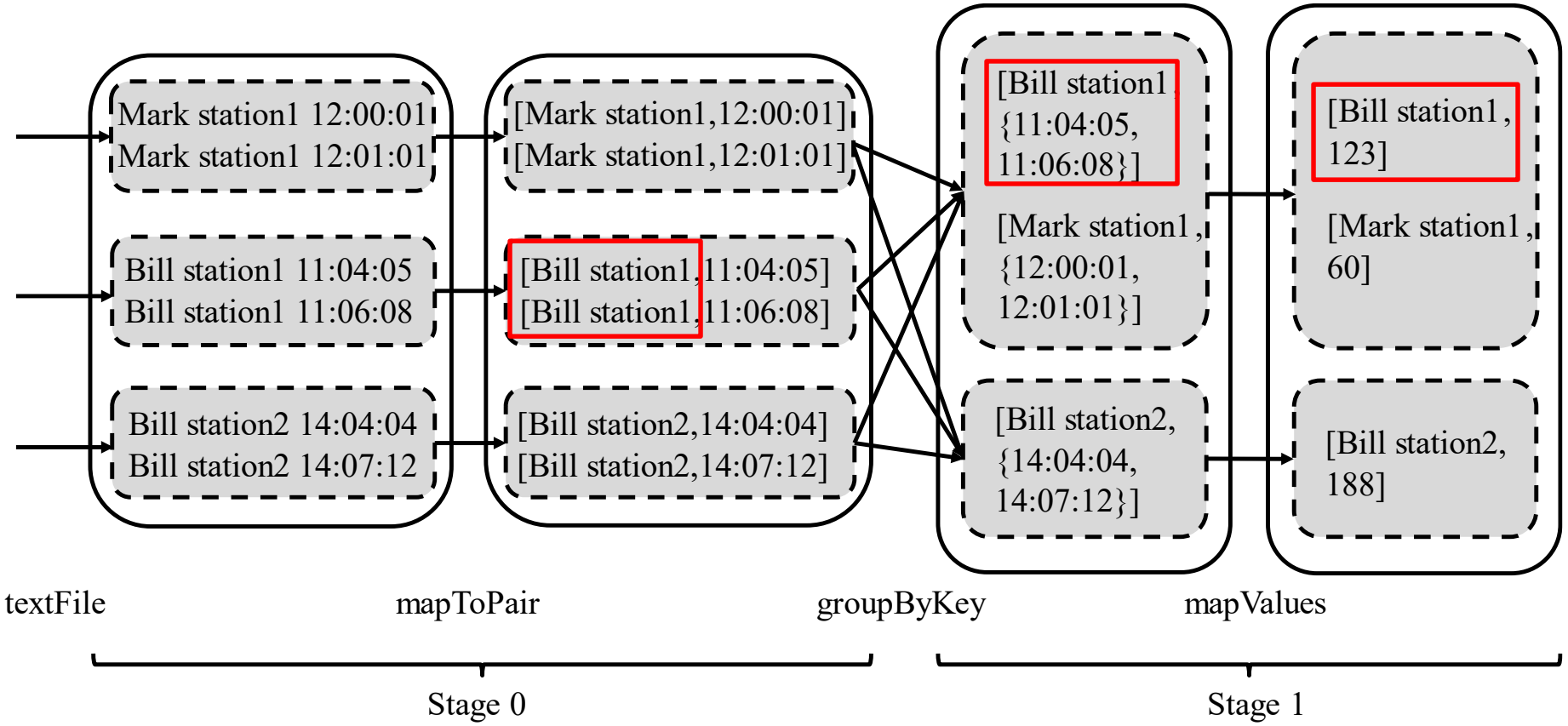
输入	输出
Mark station1 12:00:01 Mark station1 12:01:01 Bill station1 11:04:05 Bill station1 11:06:08 Bill station2 14:04:04 Bill station2 14:07:12	(Bill,station1,123) (Bill,station2,188) (Mark,station1,60)

解决方案

- mapToPair
  - 将每一行映射为[姓名 基站名, 时刻]键值对
- groupByKey
  - 按键“姓名 基站名”对键值对进行分组，得到[姓名 基站名, {时刻列表}]
- mapValues
  - 对出入基站的时刻列表进行排序后，计算每一次的停留时间并累加，得到总停留时间
- map
  - 返回(姓名, 基站名, 总停留时间)组成的三元组

reduceByKey ✗

基站统计的运行过程


Mark station1 12:00:01
Mark station1 12:01:01
Bill station1 11:04:05
Bill station1 11:06:08
Bill station2 14:04:04
Bill station2 14:07:12
[Mark station1,12:00:01]
[Mark station1,12:01:01]
[Bill station1,11:04:05]
[Bill station1,11:06:08]
[Bill station2,14:04:04]
[Bill station2,14:07:12]
[Bill station1, {11:04:05, 11:06:08}]
[Mark station1, {12:00:01, 12:01:01}]
[Bill station2, {14:04:04, 14:07:12}]
[Bill station1, 123]
[Mark station1, 60]
[Bill station2, 188]
textFile
mapToPair
groupByKey
mapValues
Stage 0
Stage 1

编写stationStatistics方法


```java
@Override
public JavaRDD<Tuple3<String, String, Integer>> stationStatistics(JavaRDD<String> lines) {
  // 将每一行输入数据映射为[姓名 基站名, 时刻]键值对
  JavaPairRDD<String, String> pairs =
      lines.mapToPair(
          new PairFunction<String, String, String>() {
            @Override
            public Tuple2<String, String> call(String line) {
              String[] splits = line.split( regex: " ");
              String groupKey = splits[0] + " " + splits[1];
              String groupValue = splits[2];
              return new Tuple2<>(groupKey, groupValue);
            }
          });

  // 按键对键值对进行分组
  JavaPairRDD<String, Iterable<String>> groups = pairs.groupByKey();

  // 计算每个人在每个基站停留的总时间
  JavaPairRDD<String, Integer> statistics =
      groups.mapValues(
          new Function<Iterable<String>, Integer>() {
            @Override
            public Integer call(Iterable<String> times) {
              // 对出入基站的时刻进行排序
              int sum = 0;
              ArrayList<String> timeList = Lists.newArrayList(times);
              Collections.sort(timeList);
              // 由于可能多次出入基站，累加停留时间
              for (int i = 0; i < timeList.size(); i += 2) {
                sum += timeToSeconds(timeList.get(i + 1)) - timeToSeconds(timeList.get(i));
              }
              return sum;
            }
          });
```

大纲

- TopK的平方和
- 广播K均值聚类
- 基站统计
- 计算传递闭包
- 网页排名前三名
- 移动平均

题目描述

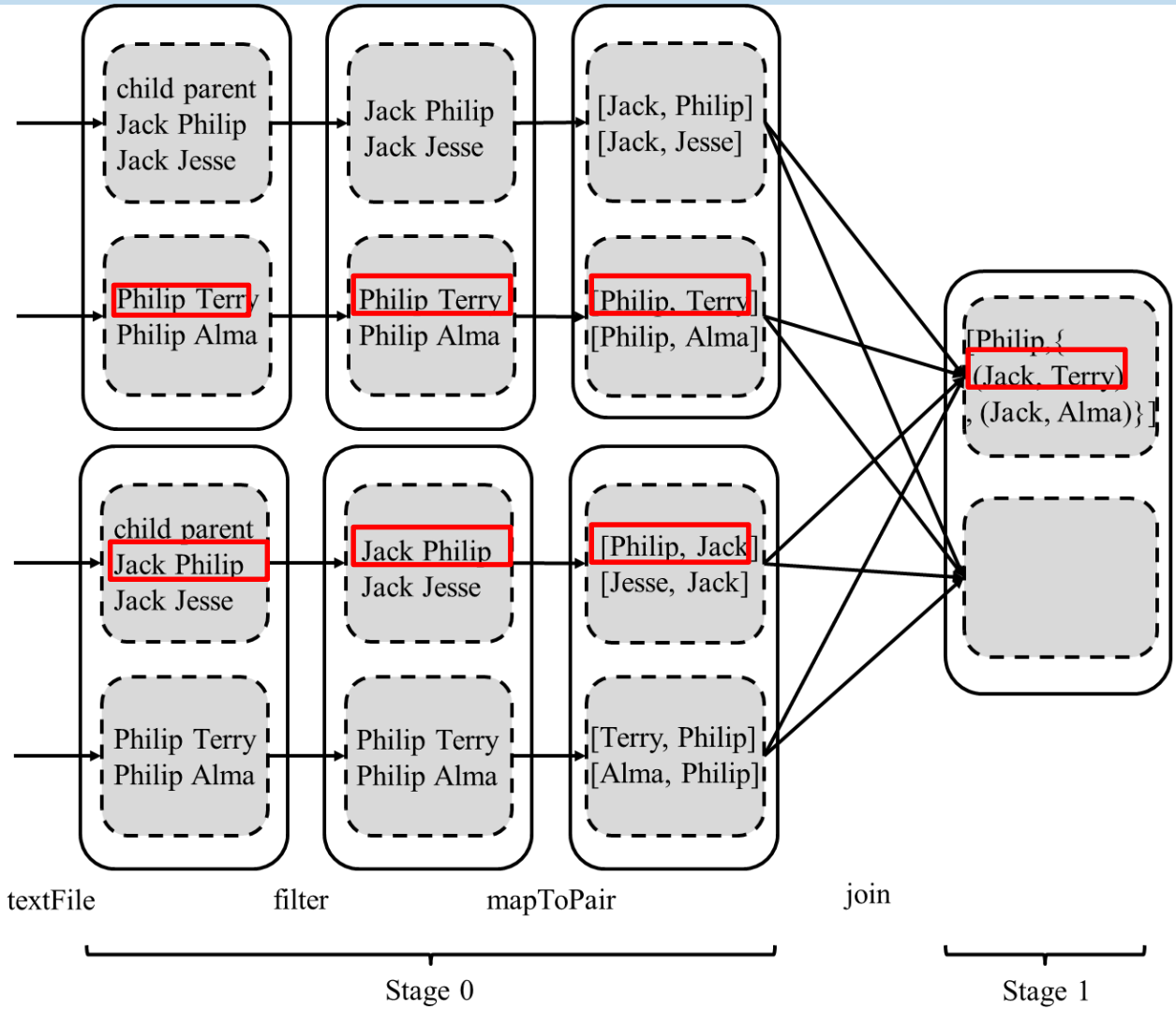
- 要求：现有一份子女-父母对应的关系文档，要求从中寻找出孙子女-祖父母关系并进行输出。
- 输入：第一行为关系标识，其余行为子女-父母的关系
- 输出：每行为一个元组，代表孙子女-祖父母的对应关系

输入	输出
child parent Jack Philip Jack Jesse Philip Terry Philip Alma	(Jack,Terry) (Jack,Alma)

解决方案

- Filter过程：
  - 过滤关系标识
- MapToPair过程：
  - 将每行文本转化成[父母, 子女]键值对
    - Jack Philip→[Philip, Jack]
  - 将每行文本转化成[子女, 父母]键值对
    - Philip Terry->[Philip, Terry]
- Join过程：
  - 进行连接操作以得到孙子女-祖父母关系
    - [Philip, {(Jack, Terry)}]

计算传递闭包运行过程


child parent
Jack Philip
Jack Jesse
Philip Terry
Philip Alma
child parent
Jack Philip
Jack Jesse
Philip Terry
Philip Alma
Jack Philip
Jack Jesse
Philip Terry
Philip Alma
Jack Philip
Jack Jesse
Philip Terry
Philip Alma
[Jack, Philip]
[Jack, Jesse]
[Philip, Terry]
[Philip, Alma]
[Philip, Jack]
[Jesse, Jack]
[Terry, Philip]
[Alma, Philip]
[Philip, {
(Jack, Terry)
, (Jack, Alma)}]
textFile
filter
mapToPair
join
Stage 0
Stage 1

编写transitive方法


```java
public class TransitiveClosuresImpl extends TransitiveClosures {

  private static final String FLAG = "child ";

  @Override
  public JavaRDD<Tuple2<String, String>> transitive(JavaRDD<String> lines) {

    // 过滤关系标识
    JavaRDD<String> association = lines.filter(
        (Function<String, Boolean>) line -> !line.contains(FLAG));

    // 将输入映射为子女-父母的键值对
    JavaPairRDD<String, String> childParentRdd = association.mapToPair(
        (PairFunction<String, String, String>)
            data -> {
              String[] token = data.split( regex: " ");
              String child = token[0];
              String parent = token[1];
              return new Tuple2<>(child, parent);
            });

    // 将输入映射为父母-子女的键值对
    JavaPairRDD<String, String> parentChildRdd = association.mapToPair(
        (PairFunction<String, String, String>)
            data -> {
              String[] token = data.split( regex: " ");
              String child = token[0];
              String parent = token[1];
              return new Tuple2<>(parent, child);
            });

    // join操作，得到[父母, {(孩子, 祖父母), (孩子, 祖父母)}]
    JavaPairRDD<String, Tuple2<String, String>> result = parentChildRdd.join(childParentRdd);

    // 返回孙子女-祖父母的关系
    return result.values();
  }
}
```

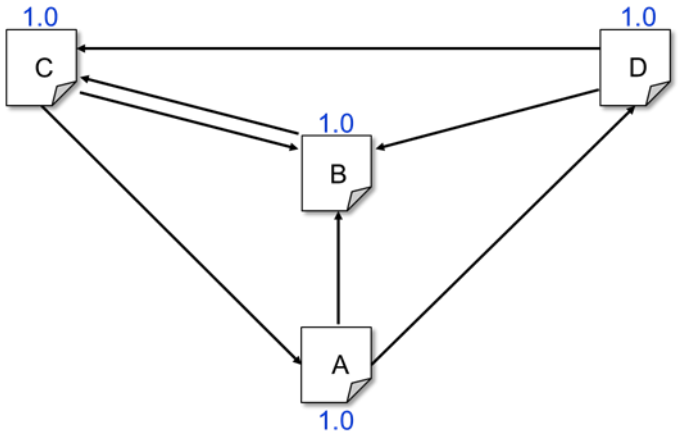
大纲

- TopK的平方和
- 广播K均值聚类
- 基站统计
- 计算传递闭包
- 网页排名前三名
- 移动平均

题目描述

- 输入：保存在文本文件中，一行为一项网页信息
  - 网页信息：(网页名 网页排名值 (出站链接 出站链接的权重...))
- 输出：前3名的网页名及其排名值

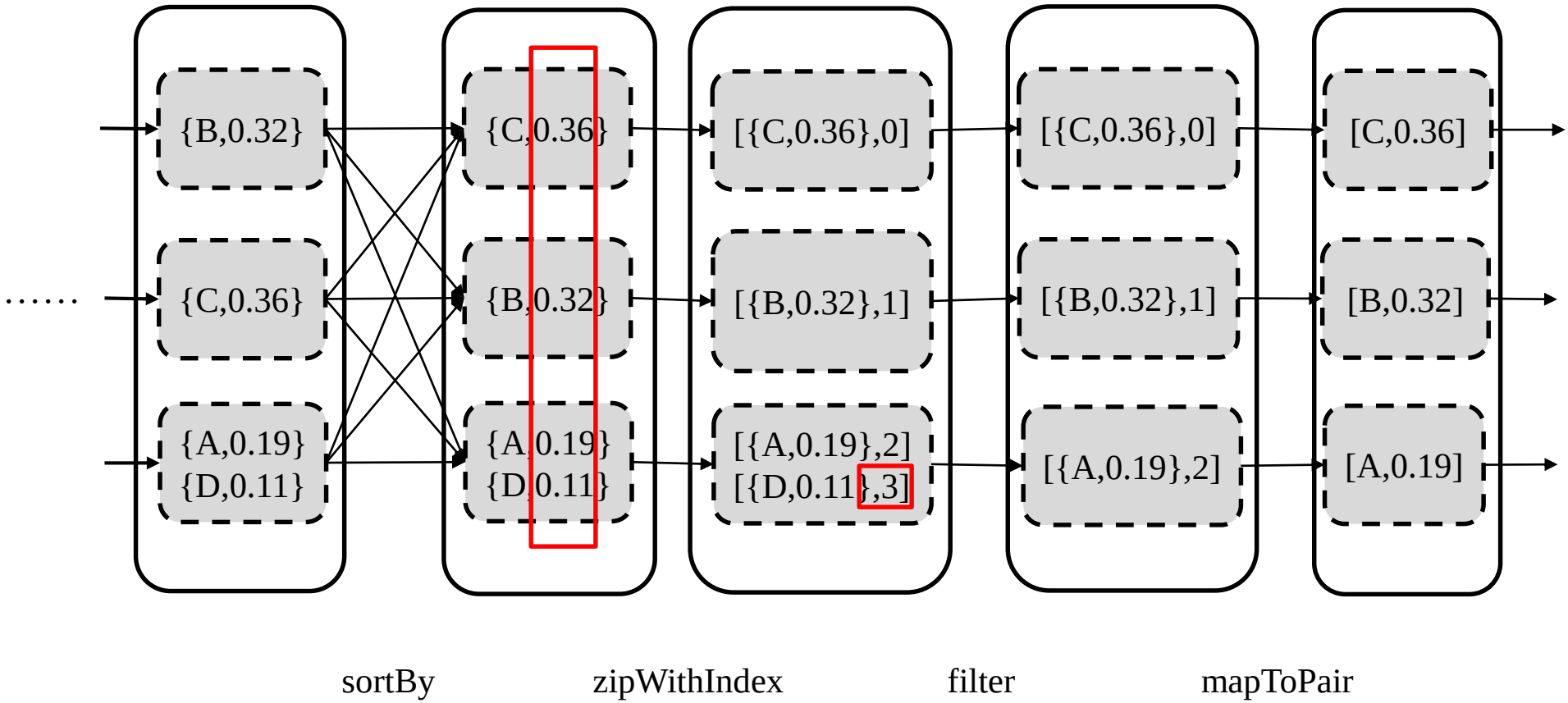
输入	输出
A 1.0 B 1.0 D 1.0 B 1.0 C 1.0 C 1.0 A 1.0 B 1.0 D 1.0 B 1.0 C 1.0	A 0.21436 B 0.36332 C 0.40833 D 0.13027



题目描述

- 算法执行过程
  - 初始时，每个网页的排名值为1
  - 每一步迭代，网页p_j对其链向的网页的排名值贡献$PR(p_j)/L(p_j)$
  - 每个网页p_i累加所有链向p_i的网页$M(p_i)$的贡献值，然后更新排名值为$0.15/N + 0.85 * \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$
  - 最终收敛
  - 排序并筛选前3名　　SortBy, ZipWithIndex,Filter

PagerankTop3的运行过程



编写PagerankTop3方法


```
// 执行迭代计算
for (int iter = 1; iter <= iterateNum; iter++) {...}

// 提取前3名
JavaPairRDD<String, Double> top3Ranks = ranks
    // 将键值对RDD转换为非键值对RDD，以便使用sortBy算子
    .map(p->new Tuple2<>(p._1, p._2)) JavaRDD<Tuple2<String, Double>>
    // 按网页排名值从高到低进行排序
    .sortBy(tuple -> tuple._2, ascending: false, ranks.getNumPartitions())
    // 将排好序的网页排名值进行编号
    .zipWithIndex() JavaPairRDD<Tuple2<String, Double>, Long>
    // 过滤，保留编号小于3的网页及其排名值
    .filter(tuple -> tuple._2 < 3)
    // 返回结果
    .mapToPair(tuple -> tuple._1);
```

大纲

- TopK的平方和
- 广播K均值聚类
- 基站统计
- 计算传递闭包
- 网页排名前三名
- 移动平均

题目描述

- 要求：在给定时间序列的情况下，计算该时间序列在 k = 2 时经过平滑移动平均转换后得到的时间序列
- 输入：时间序列键值对 [时间 t , 值 v]，该序列涵盖 1~n 区间的所有时间点且不重复
- 输出：时间序列键值对 [时间 t , 值 v]，平均值向下取整

$$V_3 = \frac{V_1 + V_2 + V_3 + V_4 + V_5}{5} = \frac{1+2+3+4+5}{5} = 3$$

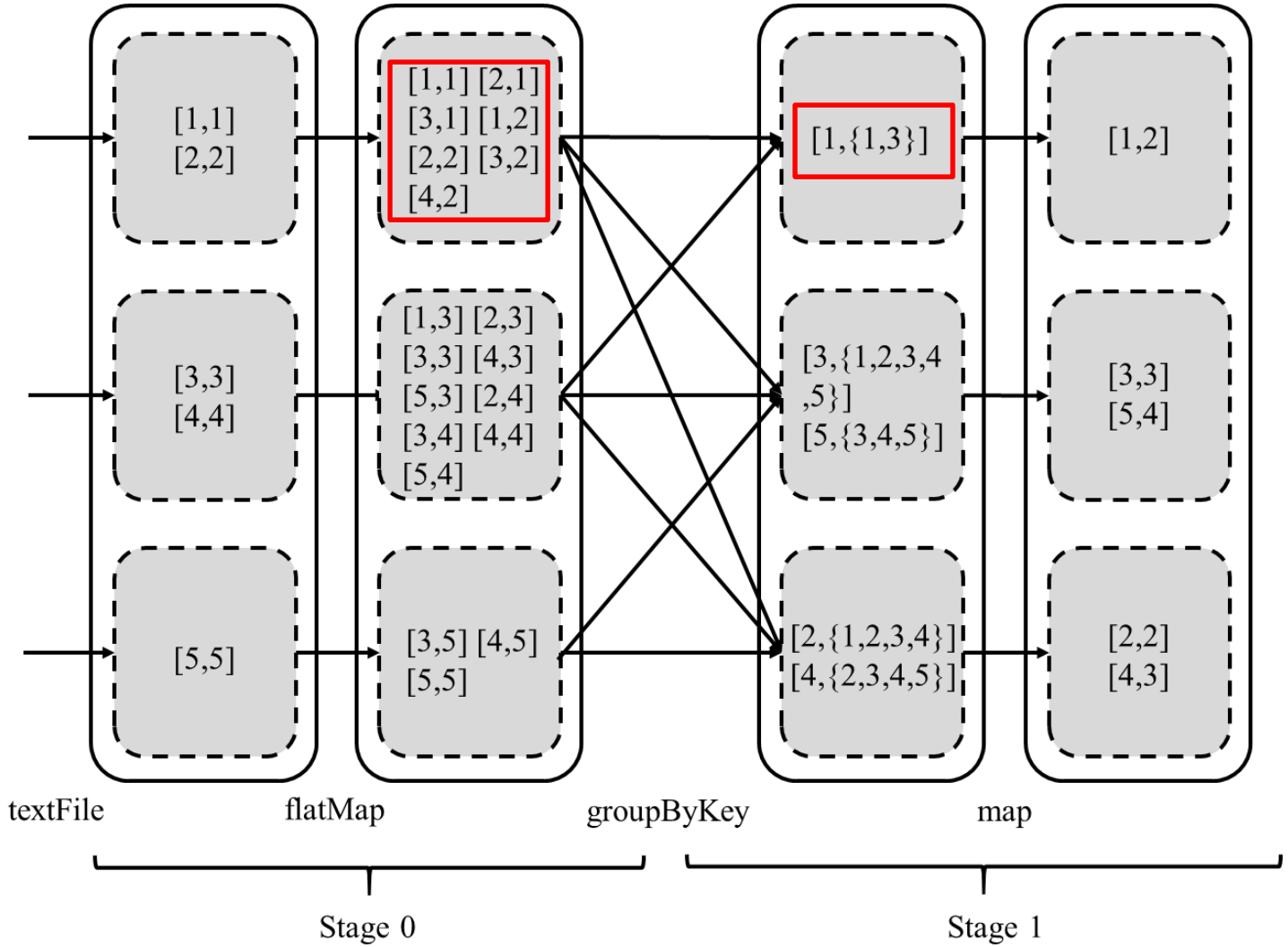
输入	输出
[1,1] [2,2] [3,3] [4,4] [5,5]	[4,3] [1,2] [3,3] [5,4] [2,2]

注：平滑移动平均在计算某一时间序列中 t 时间的值时，将 t 时间点本身以及前后各 k 个时间点的值的平均值当作 t 时间点的值

解决方案

- 作业一
  - 由于时间点边界n（即键值对数量）未知，首先统计输入RDD的数据总量以获取n
- 作业二：
  - 将t时间点的键值对转换成多个 [time,value]键值对，time为t-2~t+2中的一个的时间点，value为t时间点的值
  - 聚合某个时间点的所有键值对
  - 对各个时间点求平均值并返回最终结果[时间点，平均值]键值对

移动平均的运行过程


[1,1]
[2,2]
[3,3]
[4,4]
[5,5]
[1,1] [2,1]
[3,1] [1,2]
[2,2] [3,2]
[4,2]
[1,3] [2,3]
[3,3] [4,3]
[5,3] [2,4]
[3,4] [4,4]
[5,4]
[3,5] [4,5]
[5,5]
[1,{1,3}]
[3,{1,2,3,4
,5}]
[5,{3,4,5}]
[2,{1,2,3,4}]
[4,{2,3,4,5}]
[1,2]
[3,3]
[5,4]
[2,2]
[4,3]
textFile
flatMap
groupByKey
map
Stage 0
Stage 1

编写movingAverages方法


```java
@Override
public JavaRDD<String> movingAverages(JavaRDD<String> lines) {
  // 获取时间序列中时间点的最大值
  long n = lines.count();
  // 将t时间点的键值对转换成多个键值对[key,value]，key为t-2~t+2中的一个的时间点，value为t时间点的值
  JavaPairRDD<Integer, Integer> posAndValue =
      lines.flatMapToPair(
          (PairFlatMapFunction<String, Integer, Integer>) line -> {
            ArrayList<Tuple2<Integer, Integer>> posValue = new ArrayList<>();
            String[] posAndValue1 = line.substring(1, line.length() - 1).split( regex: ",");
            int pos = Integer.parseInt(posAndValue1[0]);
            int value = Integer.parseInt(posAndValue1[1]);
            posValue.add(new Tuple2<>(pos, value));
            for (int k = 1; k <= 2; k++) {
              int left = pos - k;
              int right = pos + k;
              if (left >= 1) {
                posValue.add(new Tuple2<>(left, value));
              }
              if (right <= n) {
                posValue.add(new Tuple2<>(right, value));
              }
            }
            return posValue.iterator();
          });
  // 聚合某个时间点的所有键值对
  JavaPairRDD<Integer, Iterable<Integer>> posIntermediateResult = posAndValue.groupByKey();
  // 对各个时间点求平均值并返回结果
  JavaRDD<String> timeSeries =
      posIntermediateResult.map(
          (Function<Tuple2<Integer, Iterable<Integer>>, String>) posTuple -> {
            int sum = 0, num = 0;
            for (Integer value : posTuple._2) {
              sum += value;
              num++;
            }
            int avg = sum / num;
            return "[" + posTuple._1 + "," + avg + "]";
          });
  return timeSeries;
}
```

谢谢！ Q&A