第九次实验报告

学号：518030910308
姓名：刘文轩

一、实验准备

1、实验环境介绍

操作系统：ubuntu 14.04
语言：Python 2
IDE：Pycharm 2019.2.3

2、实验目的

2.1　学会编写 mapper.py 和 reducer.py
2.2　实现简单的文章中的首字母统计
2.3　实现自己的 PageRank

3、实验思路

3.1 在首字母统计中，mapper.py 将单词呈现为单词与单词长度的组合，对于 reducer.py，它将通过读取 mapper.py 的输出，计算得到每个字母的平均单词长度

3.2　在 PageRank 中，mapper.py 输出的是各个页面自己的 pagerank，以及它与其它页面的关系，然后在 reducer.py 中，它根据 mapper.py 的输出，得到最后的结果

二、实验过程

1、首字母统计

1.1　ex1_mapper.py 的设计

由于我们最后的计算中需要知道每一个单词的词长，因此我们在 mapper.py 中选择输出单词与它对应的长度：

```
1.  #!/usr/bin/env python
2.
3.  import sys
4.
5.  # input comes from STDIN (standard input)
6.  for line in sys.stdin:
7.      # remove leading and trailing whitespace
8.      line = line.strip()
9.      # split the line into words
10.     words = line.split()
11.     # increase counters
12.     for word in words:
13.         # write the results to STDOUT (standard output);
14.         # what we output here will be the input for the
15.         # Reduce step, i.e. the input for reducer.py
16.         #
17.         # tab-delimited; the trivial word count is 1
18.         print '%s\t%s' % (word.lower(), len(word))
```

1.2 ex1_reducer.py 的设计

reducer.py 中的处理并不复杂，我们只需要根据单词的首字母，将其对应的放在我们需要的地方，并且按照公式计算即可。

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
current_len = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word, length = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        length = float(length)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue
    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == None:
        current_len = length
        current_count=1
        current_word = word
    else:
        if current_word[0] == word[0]:
            current_len += length
            current_count += 1
        else:
            if current_word:
            # write result to STDOUT
                print '%s\t%s' % (current_word[0], round(current_len/current_count,2))
            current_len = length
            current_count=1
```

```
            current_word = word
# do not forget to output the last word if needed!
if current_word[0] == word[0]:
    print '%s\t%s' % (current_word[0], round(current_len/current_count,2))
```

2.3 编写脚本与运行结果

为了免去一遍遍输入差不多命令的麻烦，我们编写了一个脚本，并将程序运行的结果写进了 ex1_result.txt

```
#!/bin/bash

reducer='ex1_reducer.py'
mapper='ex1_mapper.py'

hdfs dfs -rm -r temp*
hdfs dfs -mkdir tempinput
hdfs dfs -copyFromLocal test1.txt tempinput

cmd='hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.2.0.jar -files $reducer,$mapper -mapper $mapper -reducer $reducer '

input='tempinput'
output='000'
for((i=1;i<2;i++));
do
    echo "Processing_$i"
    output="tempoutput_$i"
    eval "$cmd -input $input -output $output"
    input=$output
    eval "hdfs dfs -rm -r $input/_SUCCESS"
    hdfs dfs -cat $input/* | sort
done
hdfs dfs -cat $input/* | sort  > ex1_result.txt
cat ex1_result.txt
```

我们的文本文件很简单，只有一句话：we become what we do,输出结果与课堂示例一致，实验结果良好。

```
19/11/15 07:11:31 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
b       6.0
d       2.0
w       2.67
```

2、PageRank

2.1 ex2_mapper.py 的设计

此实验中的 mapper.py，主要为 reducer.py 提供处理的“粗加工”材料。

$$PR(A) = \frac{(1-d)}{N} + d(PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n))$$

我们留意 PageRank 的公式，注意到每一个页面的 PR 不仅与指向它的页面的 PR 有关，还与指向它的页面指向的页面数量有关，这也很好理解，一个页面如果非常“专一”，那么它指向的那个页面理所应当能继承它的全部 PR，如果它很“花心”，那么它的 PR 就会被更多的页面平分。因此我们在 mapper.py 中不仅需要考虑到每个页面自身的 PR，还要考虑到它与其它页面的关系。具体代码如下：

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    line = line.strip()
    if not line:
        continue
    item = line.split()
    print '%s\t%s' % ( item[0], 0.0)
    value = float(item[1])/(len(item)-2)
    for c in item[2:]:
        print '%s\t%s' % (c, value)
    print '%s\t%s' % (item[0], '% '+' '.join(item[2:]))
```

2.2 ex2_reducer.py 的设计

我们首先检查输入的是不是网页之间的关系，如果是的话，直接保存，不要进行接下来额外的处理：

```
if '%' in data:
        links[data[0]] = data[2:]
        continue
```

然后如果输入的是网页与它的 PR，那么我们就根据公式，进行相关的计算并且添加。经过一次 mapper.py 和 reducer.py，相当于完成了一次“投票”。完整代码如下：

```
#!/usr/bin/env python
import sys
values = 0.0
alpha = 0.85
N = 4
results = dict()
links = dict()

content = sys.stdin.readlines()
flag = 0
```

```
for line in content:
    if not line.strip():
        break
    data = line.strip().split()
    if '%' in data:
        links[data[0]] = data[2:]
        continue
    page,value = data[0],float(data[1])
    if page in results:
        results[page] += value
    else:
        results[page] = value
for key in results:
    print '%s\t%f\t%s' % (key, results[key]*alpha + (1-alpha)/N, ' '.join(links[key]))
```

2.3 编写脚本与运行结果

为了反复运行 mapper.py 与 reducer.py，我们需要反复输入命令，然而在实际中这既耗费时间又没有必要，因此我们在此编写了一个脚本，用来获取结果。脚本如下：

```
#!/bin/bash

reducer='ex2_reducer.py'
mapper='ex2_mapper.py'

hdfs dfs -rm -r temp*
hdfs dfs -mkdir tempinput
hdfs dfs -copyFromLocal test2.txt tempinput

cmd='hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.2.0.jar -files $reducer,$mapper -mapper $mapper -reducer $reducer '

input='tempinput'
output='000'
for((i=1;i<20;i++));
do
    echo "Processing_$i"
    output="tempoutput_$i"
    eval "$cmd -input $input -output $output"
    input=$output
    eval "hdfs dfs -rm -r $input/_SUCCESS"
    hdfs dfs -cat $input/* | sort
done
hdfs dfs -cat $input/* | sort  > result.txt
```

```
24. cat result.txt
```

根据计算，我们要得到稳定的结果需要相对较多的迭代，然而在一段时间的迭代之后其实结果已经趋于稳定，因此在此我们使用 19 次迭代，第一次迭代之后的结果如下：

```
19/11/15 05:54:22 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
1       0.037500        2 3 4
2       0.320833        3 4
3       0.214583        4
4       0.427083        2
```

与实验预期相符。

第 19 次迭代之后的结果如下：

```
19/11/15 06:12:51 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
1       0.037500        2 3 4
2       0.373244        3 4
3       0.206760        4
4       0.382497        2
```

已经与本地多次运算的结果相当相近了。

三、实验总结

1、实验概述

本次实验的主要任务，可以总结为使用 Hadoop 进行具体的运用，需要我们自己动手设计 mapper.py 和 reducer.py，整体十分有趣，还需要编写脚本，也有一定的挑战性。

2、感想总结

在这次的实验中，学会了的东西有很多，其中最重要的就是提高了自己处理问题、收集相关资料、解决问题的能力，这在我们将来的学习和工作生活中都是很重要的。而具体细化开来，在本次实验中：

2.1 了解了 Hadoop 的更多操作

2.2 学会了编写 mapper.py 和 reducer.py 的方法

2.3 学会了统计文章中的首字母

2.4 学会了 PageRank 的原理，并且自己动手尝试编写

3、创新点

使用脚本语言编写了脚本文件来运行程序，使得代码运行的复杂而机械的指令只需要书写一次而不是不断的重复，这也为我以后的学习提供了新的方向，让我第一次认真接触到脚本这个熟悉而又陌生的名词。

4、遇到的问题

脚本编写的过程中，会遇到各种各样的问题，比如语法不熟，比如方法不对，尤其是编写循环语句时，要不断理清楚什么时候干什么事情，有的时候遇到了一些很简单的错误，但是从程序报错的信息中观察不到，这就需要我编写程序的时候更加的仔细和耐心。

```
1.  for((i=1;i<20;i++));
2.  do
3.      echo "Processing_$i"
4.      output="tempoutput_$i"
5.      eval "$cmd -input $input -output $output"
6.      input=$output
7.      eval "hdfs dfs -rm -r $input/_SUCCESS"
```

```
    hdfs dfs -cat $input/* | sort
done
```