

电类工程学导论 C 实验报告 9

518030910406 郑思榕

一、实验准备

1. 实验环境介绍

- 1) 环境：在 windows 系统中使用 VirtualBox 5.2.18 安装 Ubuntu14.04 虚拟机，从而在 UNIX 系统环境下进行本次实验。
- 2) 语言：python 2.7
- 3) 工具：本实验主要使用了分布式系统基础架构 Hadoop 和其提供的编程工具 Hadoop Streaming

2. 实验目的

- 1) 有一篇英语论文，试着写一个 mapper.py 和一个 reducer.py 来计算每个单词从“a”到“Z”的平均长度。如下图：

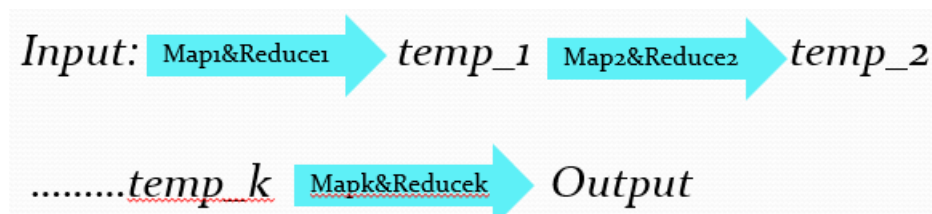
Eg. *we become what we do*

w	2.66	$= [\text{len}(\text{"we"}) + \text{len}(\text{"what"}) + \text{len}(\text{"we"})] / 3$
b	6	
d	2	

- 2) 以下是一个关于 PageRank 的基本算法: <https://en.wikipedia.org/wiki/PageRank#Algorithm>
<https://segmentfault.com/a/1190000000711128> 需要几轮迭代才能得出最终的 pagerank 值。尝试编写自己的 map .py 和 reduce .py 来实现这个算法。让它更容易理解,我给下面的例子: $\alpha = 0.85$, 结果如下:

Input:			Output:		
ID	PR	Link ID	ID	PR	Link ID
1	0.25	2 3 4	1	0.0375	2 3 4
2	0.25	3 4	2	0.3208	3 4
3	0.25	4	3	0.2146	4
4	0.25	2	4	0.4271	2

我们将通过下面这种方式得到最终的 pagerank 值:



在这个过程中，前一个 reducer 的输出成为下一个 mapper 的输入。

3. 实验原理

Hadoop Streaming 是 Hadoop 提供的编程工具，它允许用户使用任何可执行文件或者脚本文件作为 Mapper 和 Reducer。本实验通过设计 mapper.py 和 reducer.py 实现字母统计(ex1)。依据 Pagerank

的算法思路——即一个网页的排名等于所有链接到该网页的网页的加权排名之和——来设计 mapper.py, reducer.py 和批处理文件完成 pagerank 的算法实现。

二、实验过程

1. ex1

涉及文件: ex1_input.txt mapper_ex1.py reducer_ex1.py ex1_result

➤ mapper-ex1.py:

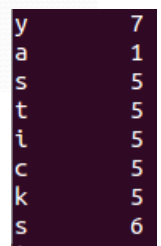
```
1  #!/usr/bin/env python
2
3  import sys
4
5  for line in sys.stdin:
6      line = line.strip()
7      words = line.split()
8      for word in words:
9          word = word.lower()
10         lenth = len(word)
11         for character in list(word):
12             if (character>='a' and character <='z'):
13                 print '%s\t%s' % (character,lenth)
```

首先设计 mapper-ex1.py 文件。mapper 和 reducer 会从输入中读取数据，一行一行处理后发送给输出端。所以 for line in sys.stdin 就是对每行数据的处理。line.strip()用于去除\n，每行里所有单词就存放在 line.split()后得到的数组里。对于每个单词，应用 word.lower()转为小写，方便后面统计字母数据。由于下图公式可知，需要计算出现某个字母的单词的长度。因此 lenth=len(word)。对于

Eg. *we become what we do*

w	2.66 = $[\text{len}(\text{"we"}) + \text{len}(\text{"what"}) + \text{len}(\text{"we"})] / 3$
b	6
d	2

word 里的每一个字母 character，如果是小写字母 a~z 就打印出来，后面带着该字母所在单词的单词长度。经过 mapper-ex1.py 后的输出如右图：



```
y 7
a 1
s 5
t 5
i 5
c 5
k 5
s 6
```

➤ reducer-ex1.py

reducer-ex1.py 将 mapper-ex1.py 的输出作为自己的输入。根据上面的公式，只需分别统计出现 a~z 的单词总长度/单词个数，即是所求的答案。reducer-ex2.py 的完整代码如下：

```
1  #!/usr/bin/env python
2
3  from operator import itemgetter
4  import sys
5
6  current_character = None
7  current_count = 0
8  character = None
9  numofword = 0
```

```

10
11 for line in sys.stdin:
12     line = line.strip()
13
14     character,count = line.split('\t',1)
15
16     try:
17         count = int(count)
18     except ValueError:
19         continue
20
21     if current_character == character:
22         numofword += 1
23         current_count += count
24     else :
25         if current_character:
26             print '%s\t%s' % (current_character,current_count/float(numofword))
27             current_count = count
28             current_character = character
29             numofword = 1
30
31 if current_character == character:
32     print '%s\t%s' % (current_character,current_count/float(numofword))

```

由于 mapper-ex1.py 的输出自动根据字母顺序就进行了排序，因此一旦 `current_character != character`，则代表着该字母统计结束，就可以进行 print 输出。`current_count` 指出现当前统计的字母的所有单词的单词长度总和，`numofword` 指单词个数。最后输出时需用 `float` 类型转换来输出小数。

➤ 结果展示(文件夹 ex1_result)

a	5.66113507426
b	6.18282380397
c	7.48196366498
d	6.02416413068
e	6.12404729334
f	4.74471707191
g	7.16986952205
h	4.84363666224
i	6.17924665503
j	5.62189054726
k	6.0793377592
l	6.70853140305
m	6.136282409
n	6.27280568802
o	5.41913675426
p	7.17320632701
q	7.53239017126
r	6.70745961779
s	6.41378111886
t	5.60670842634
u	6.53741194578
v	6.64724361573
w	5.21103208232
x	7.43237704918
y	5.6067520846
z	6.92863762743

2. ex2

涉及文件: ex2_input.txt mapper-ex2.py reducer-ex2.py ex2_result

➤ mapper-ex2.py

```
1  #!/usr/bin/env python
2
3  import sys
4
5  try:
6      class Page:
7          def __init__(self, id, pk):
8              self.id = id
9              self.pk = pk
10
11         d = 0.85
12         N = 0 # total num of pages
13         Pagelist = []
14         linkrelation = {}
15
16         for line in sys.stdin:
17             N += 1
18             line = line.strip()
19             page_id, page_pk, strlinkid = line.split(None, 2)
20             page_id = int(page_id)
21             page_pk = float(page_pk)
22             Pagelist.append(Page(page_id, page_pk))
23             linkrelation[page_id] = strlinkid
24
25         # initial pagerank and relationship of pages
26         for page in Pagelist:
27             listlinkid = linkrelation[page.id].split()
28             print '%s=%s;%s:%s' % (str(page.id),0,page.id,linkrelation[page.id])
29             length = len(listlinkid)
30             for i in listlinkid:
31                 print '%s=%s;%s:%s' % (str(i), page.pk / float(length),page.id,
32                                         linkrelation[page.id])
33
34         #2=0.8333;1:2 3 4
35     except Exception , Argument:
36         print str(Argument)
```

根据 Wiki 中针对 Pagerank 的算法描述, 网页 A 的 page 可以表示为:

$$PR(A) = \frac{1-d}{N} + d \left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \dots \right)$$

其中 d 是打开该网页上链接(而不是新开一个网页)的概率, 一般设置为 0.85。N 是所有网页数, 本次实验共有 4 个网页, 所以 N=4。PR(B)/L(B)是指, 在网页 B 链接到网页 A 的前提下, (B 的 pagerank)/(B 中所有的链接数量), 如本实验第一次迭代输入 $\begin{matrix} 2 & 0.25 & 3 & 4 \end{matrix}$, 则 PR(2)/L(2) = 0.25/2=0.125。所以 mapper 的输出应是所有网页上所有链接的 PR(B)/L(B)的值。首先定义 Page 类型, 包含 Page.id 和 Page.pk 便于存储, 列表 Pagelist 用来存放所有的 Page。然

后进行数据的读取和处理。每行输入数据的格式为 `2 0.25 3 4`，所以 `line.split(None,2)` 以空格进行划分，划分两次，第一个元素是 `page_id`，第二个元素是 `page_pk`，剩下的是该网页上的所有链接 `strlinkid`。然后进行数据类型转换，`page_pk` 转换成 `float` 类型，并生成 `Page` 类型节点存放到 `Pagelist` 中。而链接关系也存放到字典 `linkrelation` 里。

1	0.25	2 3 4
2	0.25	3 4
3	0.25	4
4	0.25	2

然后对于 `Pagelist` 中的网页进行 $PR(B)/L(B)$ 的计算和输出。观察到输入数据(上图)中，没有网页链接到网页 1，所以对于每个网页应先输出 $PR(B)/L(B)=0$ 的情况，见代码第 28 行。然后将 `strlinkid` 转换成列表 `listlinkid`，对于其中每个 `page`， $PR(B)/L(B)=page.pk/float(length)$ 。注意应将链接关系也打印出来，因为 `reducer-ex2.py` 需要。最后输出格式类似为：`2=0.8333;1:2 3 4`，意为“网页 2=网页 2 的 PageRank；链接到网页 2 的父网页：父网页的所有链接”。在本地 terminal 只运行 `mapper-ex2.py` 后的输出如下图：

```
hduser@zsir-VirtualBox:~/experiment/src$ cat ex2_input.txt | ~/experiment/src/mapper-ex2.py
1=0;1:2 3 4
2=0.08333333333333333;1:2 3 4
3=0.08333333333333333;1:2 3 4
4=0.08333333333333333;1:2 3 4
2=0;2:3 4
3=0.125;2:3 4
4=0.125;2:3 4
3=0;3:4
4=0.25;3:4
4=0;4:2
2=0.25;4:2
```

➤ reducer-ex2.py

```
1  #!/usr/bin/env python
2
3  from operator import itemgetter
4  import sys
5
6  try:
7      N = 4
8      d = 0.85
9      totalpk = 0
10     current_page = None
11     current_pk = 0
12     pk_dic = {}
13     linkrelation = {}
14     page_id = None
15     linkid = None
16
```

```

17 # 2=0.8333;1:2 3 4
18 for line in sys.stdin:
19     line = line.strip()
20     if line:
21         PageandRank, PageandRelation = line.split(';')
22
23         Pageid_in_relation, linkid = PageandRelation.split(':')
24         linkrelation[Pageid_in_relation] = linkid
25
26         page_id, page_pk = PageandRank.split('=')
27         page_pk = float(page_pk)
28
29         if page_id in pk_dic:
30             pk_dic[page_id].append(page_pk)
31         else:
32             pk_dic[page_id] = []
33             pk_dic[page_id].append(page_pk)
34
35     for page in pk_dic.keys():
36         for pk in pk_dic[page]:
37             totalpk += pk
38         finalpk = (1 - d) / N + d * (totalpk)
39         print '%s\t%s\t%s' % (page, finalpk, linkrelation[page])
40         totalpk = 0
41
42 except Exception, e:
43     print e

```

首先对 reducer.py 的数据进行数据处理。对于每行输入信息进行 split(';')，第一个元素是网页及由 mapper-ex2.py 计算得到的 PR(B)/L(B)值，第二个元素是链接关系 PageandRelation。对 PageandRelation 进行 split(':') 拆分，加入到链接关系字典 linkrelation，键值对关系为“网页 id”=>“该网页上的所有链接”。对 PageandRank 进行 split('=') 拆分，加入到 pageid 和 PageRank 的字典 pk_dic 中，键值对关系为“网页 id”=>“由 mapper 得到的该网页所有 PR(B)/L(B)值组成的列表”，注意字典 pk_dic 中的值是一个列表。

然后对 pk_dic 中的所有 PR(B)/L(B)进行求和得到 totalpk，最后根据 PageRank 的公式进行 finalpk 的计算，打印出结果即可。注意输出格式应与 mapper-ex2.py 的输入格式一致。

第一次运行 mapper-ex2.py 和 reducer-ex2.py 后的输出结果如下：

```

hduser@zslr-VirtualBox:~/experiment/src$ cat ex2_input.txt | ~/experiment/src/mapper-ex2.py | sort -k1,1 | ~/experiment/src/reducer-ex2.py
1      0.0375  2 3 4
3      0.214583333333  4
2      0.320833333333  3 4
4      0.427083333333  2

```

➤ batch_ex2.sh

因为需要多次迭代，故创建脚本文件 batch_ex2.sh（如下图）

input = \$output 每次运行把上一次迭代的输出作为这次的输入，然后每次把输入文件 \$input 删除最后在本地图建文件夹 ex2_result，将最后一次的输出结果保留在本地图


```

#/bin/bash

command='hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar -files mapper-ex2.py,reducer-ex2.py
mapper mapper-ex2.py -reducer reducer-ex2.py'
rm='hadoop fs -rm -r '
cp2local='hadoop fs -copyToLocal '
input='ex2_input'
for ((i=1;i<$1+1;i++));
do
    echo "Processing $i"
    output="ex2_output$i"
    eval "$command -input $input -output $output"
    input=$output
    eval "$rm $input/_SUCCESS"
done
mkdir /home/hduser/ex2_result
eval "$cp2local $output/* /home/hduser/ex2_result"

```

- 结果展示(ex2_output 文件夹)
- 在经过 5 次迭代后得到输出如下图，当迭代次数越多，输出趋于一个稳定值。

1	0.0375	2	3	4
3	0.205733649088	4		
2	0.380087740885	3	4	
4	0.376678610026	2		

三、实验总结

1. 实验概述

本实验通过设计 mapper.py、reducer.py 和批处理文件来实现字母统计和 PageRank 算法。

2. 实验心得

本实验我收获良多，主要有以下几点：

- 1) 学会了 Hadoop 的基本操作，如 copyFromLocal, cat, 运行 Hadoop streaming 等
- 2) 学会了如何设计 mapper.py 和 reducer.py 来实现 Hadoop streaming
- 3) 学会了 PageRank 算法与如何通过 Hadoop streaming 实现
- 4) 学会了如何编写 linux 批处理文件

3. 遇到的困难及解决方案

- 1) ex2 中在本地运行 mapper.py 正常输出，在 Hadoop 云端运行就报错

Solution: 通过在 mapper.py 各个地方 print 出各种变量以及查看 Hadoop stderr, 得知是 mapper.py 自动将输入文件分成两部分，第一部分只包含 page 1 和 page 2 里的链接，而 page 1 中有指向 page 3 的链接，因此按我以前的算法需要 print 出 page1 的子网页中的链接，即 page 3 中的链接，但找不到信息因此报错。所以我修改算法，不 print 出子网页中的链接，反而 print 出父网页中的链接，即 page 1 的链接，在 reducer.py 中再统一收集所有网页的链接关系，因此不报错。

4. 实验创新点

- 1) 不采用给出的 reducer.py 的算法结构，不设置变量 current_page 和 current_pk。创建自己的 reducer.py 结构，即创建 pk_dic 字典，里面放 page_id=>[page_pk1, page_pk2,...]的键值对，最后对列表里的所有 page_pk 进行求和。

最后，衷心感谢实验中老师和各位助教的帮助！