**2018年12月**



**大数据计算基础**

题 目：Hadoop环境下实现基于TF-IDF算法的搜索引擎和分布式PageRank算法

专 业： 软件工程

学 号： 1163710228

姓 名： 刘思琦

课程类别： 选修

# 1. 问题描述

**在此处写清你的题号、题目名称、复制你的问题描述。**介绍你要解决的问题

a. 问题的背景/意义

b. 问题的技术难点

**题号：**选修（3学分）题目第2题

**题目名称：**Hadoop环境下实现基于TF-IDF算法的搜索引擎和分布式PageRank算法

**问题描述：**

现有一组“纽约时报”评论的数据，数据来源如下：

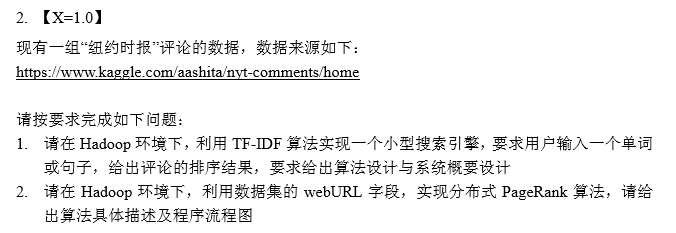
https://www.kaggle.com/aashita/nyt-comments/home

请按要求完成如下问题：

1.请在Hadoop环境下，利用TF-IDF算法实现一个小型搜索引擎，要求用户输入一个单词或句子，给出评论的排序结果，要求给出算法设计与系统概要设计。

2.请在Hadoop环境下，利用数据集的webURL字段，实现分布式PageRank算法，请给出算法具体描述及程序流程图。

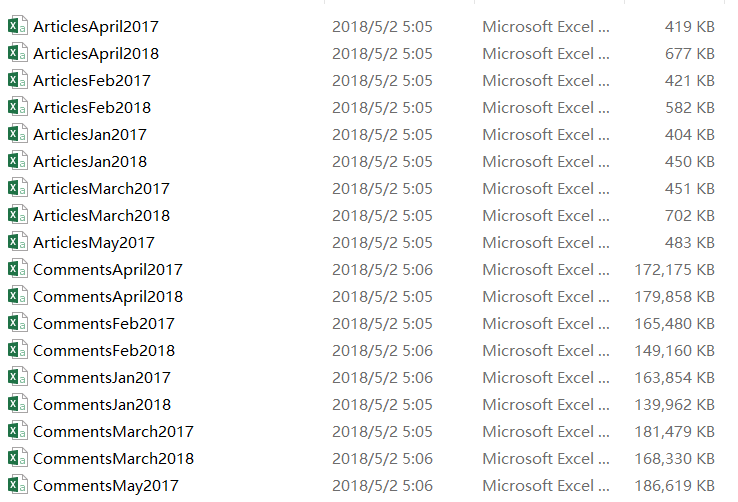
原题目粘贴如下：



**待解决问题的介绍：**

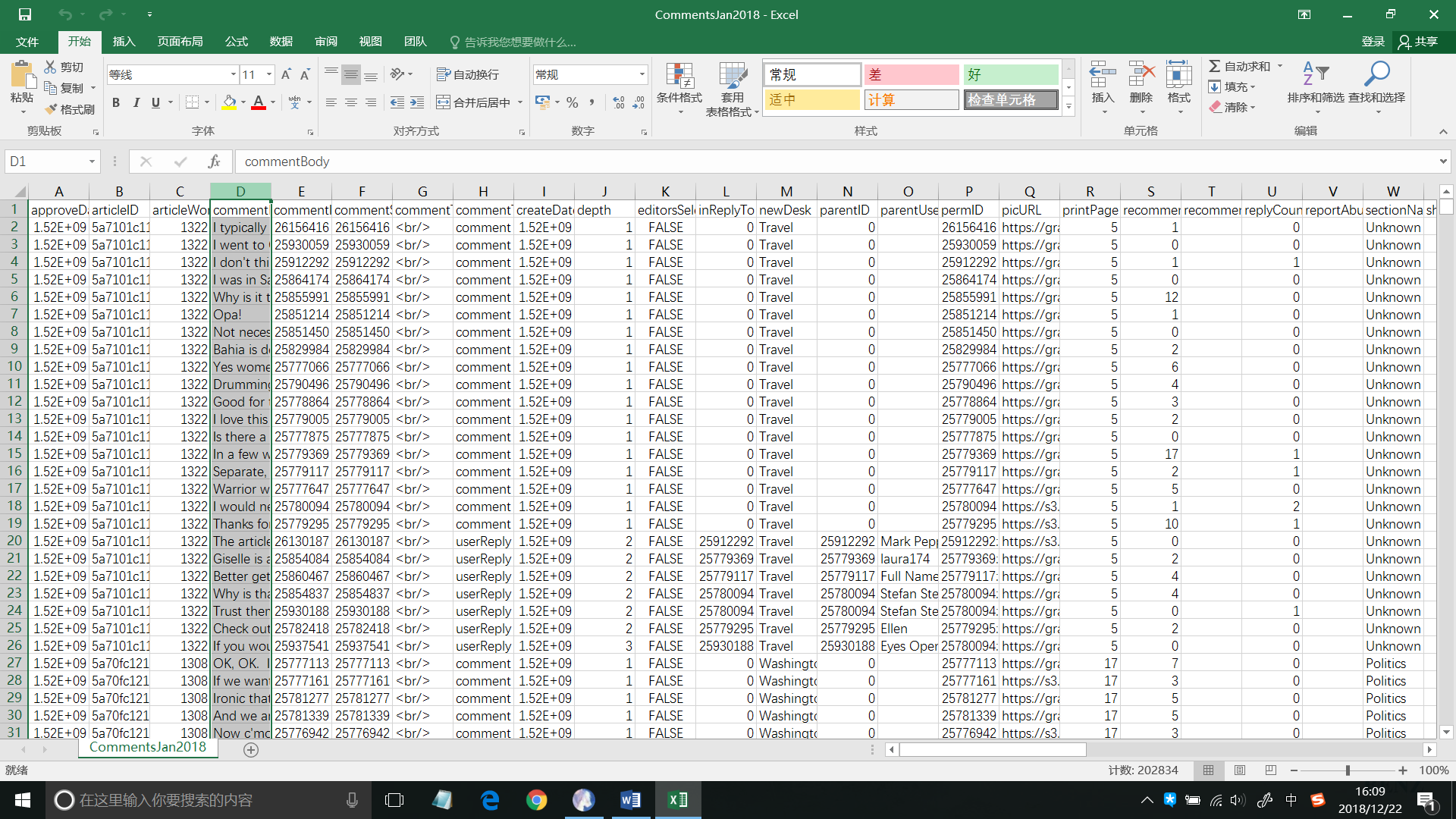
**问题的第1问**要求利用TF-IDF算法实现对数据集的一个小型搜索引擎，针对用户的输入对数据集中的数据的相关性进行排序并输出排序结果。

访问数据来源的网址，下载数据之后查看数据，我们可以发现数据集分为两部分，分别是“纽约时报”的文章统计和评论统计，如图所示是下载好的数据集：



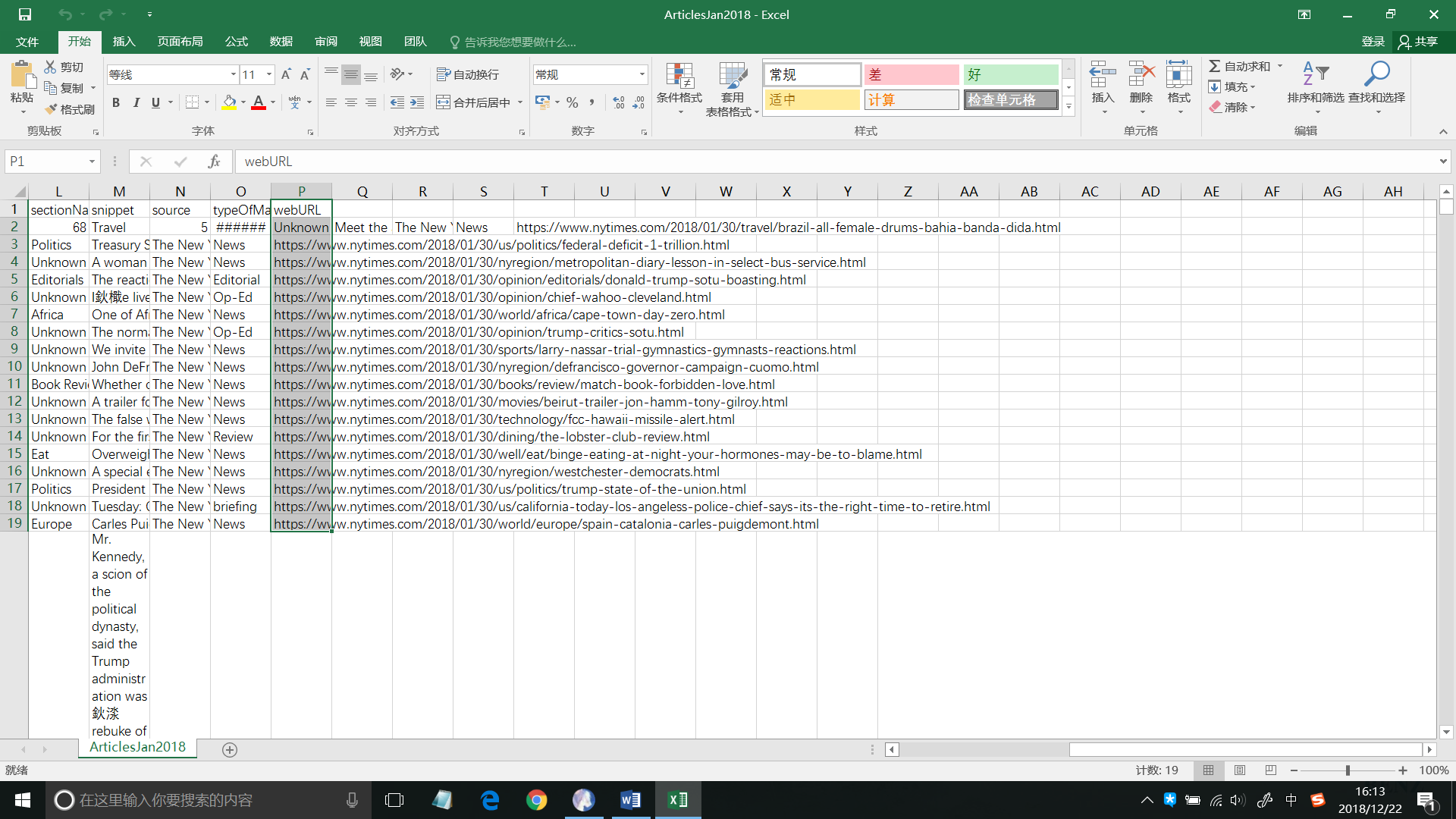
第1问要求是根据用户的输入给出评论的排序结果，根据题意可知，我们应该采用数据集中的comments数据作为程序的输入。

以CommentsJan2018为例，查看文件：



每一条评论都对应一个commentBody，分析可知，我们只需要基于commentBody部分实现TF-IDF算法即可。

**问题的第2问**要求我们在Hadoop环境下利用数据集的webURL字段，实现分布式PageRank算法。检索数据集后可以发现，webURL字段只出现在了articles数据中，以ArticlesJan2018文件为例：



每一个文章都对应一个webURL字段，基于此字段内容实现PageRank算法即可。

**问题的背景/意义：**

问题的第1问的背景意义在于，利用TF-IDF算法对“纽约时报”的评论数据进行整理之后，可以使用户更方便的对评论内容进行检索，最终的程序可以实现根据用户的输入搜索到具有相关性的全部评论内容，方便用户对评论的查阅。

问题的第2问的背景意义在于，利用分布式PageRank算法对“纽约时报”的文章数据集的webURL字段进行整理之后，可以得出每一个webURL字段的PageRank值，也就是这个网页对整个数据集中的网页字段的“重要性”程度，可以起到一个大数据舆情分析的作用。

# 2. 基于的系统/算法

介绍你使用的主要的系统或者算法库

a. 系统架构/算法原理

b. 你为什么选择使用这个系统/算法

c. 你是如何使用这个系统/算法的

**总述：在问题的第一问中，我使用的算法是基于Java实现的TF-IDF算法，在算法中根据数据的实际情况应用了Java中的IKAnalyzer分词器；在问题的第二问中，我使用的是基于Java实现的PageRank算法和基于Python实现的HTTPs爬虫算法。下面我将对这几种算法依次进行详细的介绍，并阐述算法的部分踩坑历程及算法的可行性，以及对最终的算法选择作出解释。**

**一.基于Java实现的TF-IDF算法**

**a.系统架构/算法原理**

**（1）概念：**

TF-IDF（term frequency-inverse document frequency）是一种用于资讯检索与资讯探勘的常用加权技术。TF-IDF是一种统计方法，用以评估一字词对于一个文件集或者一个语料库中的其中一份文件的重要程度。**字词的重要性随着它在文件中出现的次数成正比增加，但是同时也会随着它在语料库中出现的频率成反比下降。**TF-IDF加权的各种形式常被搜索引擎采用，作为文件与用户查询之间相关程度的度量或评级

**（2）原理：**在一份给定的文件中，**词频（term frequency，TF）**指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化（分子一般小于分母 区别于IDF），以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件里有更高的词频，而不管该词语重要与否）。

**逆向文件频率（inverse document frequency，IDF）**指的是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

**TFIDF的主要思想是**：如果某个词或短语在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。TFIDF实际上是：TF \* IDF，TF词频(Term Frequency)，IDF反文档频率(Inverse Document Frequency)。**TF**表示词条在文档d中出现的频率（另一说：**TF词频(Term Frequency)**指的是某一个给定的词语在该文件中出现的次数）。IDF的主要思想是：如果包含词条t的文档越少，也就是n越小，IDF越大（见后续公式），则说明词条t具有很好的类别区分能力。如果某一类文档C中包含词条t的文档数为m，而其它类包含t的文档总数为k，显然所有包含t的文档数n = m + k，当m大的时候，n也大，按照IDF公式得到的IDF的值会小，就说明该词条t类别区分能力不强。（另一说：**IDF反文档频率(Inverse Document Frequency)**是指果包含词条的文档越少，IDF越大，则说明词条具有很好的类别区分能力。）但是实际上，有时候，如果一个词条在一个类的文档中频繁出现，则说明该词条能够很好代表这个类的文本的特征，这样的词条应该给它们赋予较高的权重，并选来作为该类文本的特征词以区别与其它类文档。这就是IDF的不足之处。

在一份给定的文件里，**词频**（term frequency，TF）指的是某一个给定的词语在该文件中出现的频率。这个数字是对**词数**(term count)的归一化，以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词数，而不管该词语重要与否。）对于在某一特定文件里的词语t_{i}来说，它的重要性可表示为：

\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}

以上式子中n_{i,j}是该词t_{i}在文件d_{j}中的出现次数，而分母则是在文件d_{j}中

所有字词的出现次数之和。

**逆向文件频率**（inverse document frequency，IDF）是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到：

\mathrm{idf_{i}} =  \log \frac{|D|}{|\{j: t_{i} \in d_{j}\}|}

其中：

|D|：语料库中的文件总数。

|\{ j: t_{i} \in d_{j}\}|：包含词语t_{i}的文件数目（即n_{i,j} \neq 0的文件数目）如果该词语不在语料库中，就会导致被除数为零，**因此一般情况下使用1 + |\{j : t_{i} \in d_{j}\}|**

然后：\mathrm{tf{}idf_{i,j}} = \mathrm{tf_{i,j}} \times  \mathrm{idf_{i}}

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

**举一个例子可以浅显的说明TF-IDF算法的实际应用：**

假如一篇文件的总词语数是100个，而词语“电商”出现了3次，那么“电商”一词在该文件中的词频就是3 / 100 = 0.03。如果“电商”一词在1000份文件出现过，而语料库中总文件数是10000000的话，其逆向文件频率就是log(10000000 / 1000) = 4。最后的TF-IDF的分数为0.03 \* 4 = 0.12。

**（3）关键词与文件相关性计算**

根据关键字k1，k2，k3进行搜索的文件相关性计算为：TF1 \* IDF1 + TF2 \* IDF2 + TF3 \* IDF3。假设文档A的总词量为1000，k1，k2，k3三个词在文档A中分别出现了100，200，50次，包含k1，k2，k3三个词的文档分别有1000，10000，5000，语料库中总文档数为10000。根据以上数据计算：

TF1 = 100 / 1000 = 0.1，

TF2 = 200 / 1000 = 0.2，

TF3 = 50 / 1000 = 0.05；

IDF1 = log(10000 / 1000) = 2.3，

IDF2 = log(10000 / 10000) = 0，

IDF3 = log(10000 / 5000) = 0.69；

那么关键字k1，k2，k3与文档A的相关性 = 0.1 \* 2.3 + 0.2 \* 0 + 0.05 \* 0.69 = 0.2645。

在某个一共有1000词的网页中，“大数据”、“的”、“应用”分别出现了 2 次、35 次 和 5 次，那么它们的词频就分别是 0.002、0.035 和 0.005。这三个数相加得0.042，就是查询“大数据的应用”与相应网页之间相关性的一个简单的度量。概括地讲，如果一个查询包含关键词 w1,w2,...,wN, 它们在一篇特定网页中的词频分别是: TF1, TF2, ..., TFN。 那么，这个查询和该网页的相关性就是:TF1 + TF2 + ... + TFN。基于上面的例子，词语“的”占了总词的 80% 以上，而它对确定网页的主题几乎没有用。我们称这种词叫“应删除词”（Stopwords)，也就是说在度量相关性时不应考虑它们的频率。在汉语中，应删除词还有“是”、“和”、“中”、“地”、“得”等等几十个。忽略这些应删除词后，上述网页的相似度就变成了0.007，其中“大数据”词贡献了 0.002，“应用”贡献了 0.005。在汉语中，“应用”是个很通用词，而“大数据”是个很专业词，后者在相关性计算中比前者重要。因此我们需要给汉语中的每一个词给一个权重，这个权重的设定必须满足下面两个条件：

1. 一个词预测主题能力越强，权重就越大，反之，权重就越小。我们在网页中看到“大数据”这个词，能基本了解网页的主题。而我们看到“应用”这个词，对主题基本上还是一无所知。因此，“大数据“的权重就大。

2. 应删除词的权重应为零。如果一个关键词只在很少的网页中出现，我们通过它就容易锁定搜索目标，它的权重也就应该大。反之如果一个词在大量网页中出现，如“的”等应删词，我们看到它仍然不很清楚要找什么内容。概括地讲，假定一个关键词 ｗ 在M个网页中出现过，那么M越大，ｗ的权重越小，反之亦然。在信息检索中，使用最多的权重就是“逆文本频率指数” （Inverse document frequency 缩写为IDF），它的公式为log（Ｄ／Ｄｗ）其中Ｄ是全部网页数，Dw是包含关键词w的网页数。比如，我们假定中文网页数是Ｄ＝１０亿，应删除词“的”在所有的网页中都出现，即Ｄｗ＝1０亿，那么它的IDF ＝ log(10亿 / 10亿）= ０。假如专用词“大数据”在200万个网页中出现，即Ｄｗ＝２００万，则它的权重IDF ＝ log(500) = 6.2。又假定通用词“应用”，出现在5亿个网页中，它的权重ＩＤＦ = log(2)，则只有 0.7。也就只说，在网页中找到1个“大数据”的比重相当于找到9个“应用”的比重。利用 IDF，上述相关性计算个公式就由词频的简单求和变成了加权求和，即 TF1 \* IDF1 +　TF2 \* IDF2 ＋... + TFN \* IDFN。在上面的例子中，该网页和“大数据的应用”的相关性为 0.0161，其中“大数据”贡献了 0.0126，而“应用”只贡献了0.0035。

**b.我为什么选择使用这个系统/算法**

TF-IDF算法非常适用于“衡量关键字在文本中的重要性”这一类问题，应用此算法可以很好地解决问题的要求，即基于用户的输入给出评论的排序结果。

**c.我是如何使用这个系统/算法的**

此算法在程序中的应用主要包括**分词、统计和计算三部分**，首先针对评论的id对应的评论内容进行分词；然后统计每个评论id对应的内容的总词数、语料库的总内容数、包含指定词的评论数；然后进行TF-IDF值的计算并输出，将输出的文件下载到本地之后用户可以进行输入、检索等一系列操作。

**二. Java IKAnalyzer分词器**

**a. 系统架构/算法原理**

IKAnalyzer 是一个开源的，基于Java语言开发的轻量级的中文分词工具包。新版本的IKAnalyzer3.0发展为面向Java的公用分词组件，独立于Lucene项目，同时提供了对Lucene的默认优化实现。

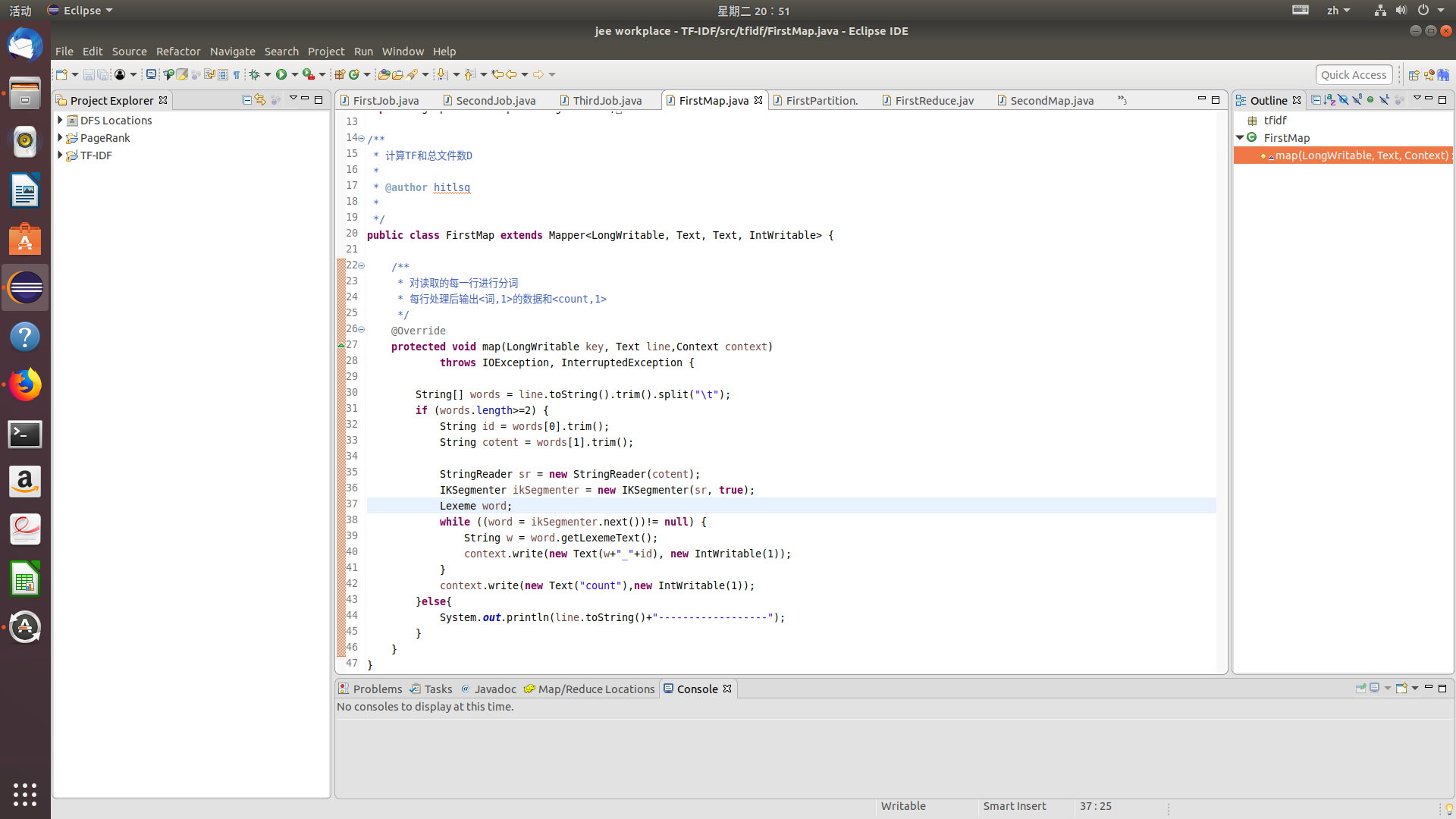
使用IKAnalyzer进行中文分词的基本原理是把文本和词典中的词进行比较，按照词典中已有的词进行匹配，匹配完成之后即达到了分词的效果。将一段文字进行IK分词处理一般经过：**词典加载、预处理、分词器分词、歧义处理、善后结尾**五个部分。

**b. 你为什么选择使用这个系统/算法**

**因为题目1要求根据用户的输入进行相关性的评论排序结果，而用户输入的一般为关键字或者关键词，所以对评论的文本集进行中文分词之后可以使用TF-IDF算法得出关键字词的词频，最后经由用户的检索，就可以输出每一条评论中用户输入的关键字/关键词出现的次数。**

**c. 你是如何使用这个系统/算法的**

首先把数据集每一行都整理成id + 制表符 + 内容的形式，然后按行进行读取，先读取id后读取内容，分别存储，然后对内容进行分词。



**三. 基于Java实现的PageRank算法**

**a. 系统架构/算法原理**

**PageRank，**网页排名，又称网页级别，传说中是PageRank算法拯救了谷歌，它是根据页面之间的超链接计算的技术，作为网页排名的要素之一。它通过网络浩瀚的超链接关系来确定一个页面的等级。Google把从A页面到B页面的链接解释为A页面给B页面投票，根据投票的来源(甚至来源的来源，即链接到A页面的页面）和投票目标的等级来决定新的等级。简单地说，一个高等级的页面可以使其他低等级页面的等级提升。

**PageRank的基本思想**：

对网页的重要程度进行排序，也就是网络中各个节点的重要程度。如果网页T存在一个指向网页A的连接，则表明T的所有者认为A比较重要，从而把T的一部分重要性得分赋予A。这个重要性得分值为：PR（T）/L(T)。

其中PR（T）为T的PageRank值，L(T)为T的出链数。

则A的PageRank值为一系列类似于T的页面重要性得分值的累加。

即一个页面的得票数由所有链向它的页面的重要性来决定，到一个页面的超链接相当于对该页投一票。一个页面的PageRank是由所有链向它的页面（链入页面）的重要性经过递归算法得到的。一个有较多链入的页面会有较高的等级，相反如果一个页面没有任何链入页面，那么它没有等级。

**PageRank的计算方式：**

假设一个由只有4个页面组成的集合：A，B，C和D。如果所有页面都链向A，那么A的PR（PageRank）值将是B，C及D的和。

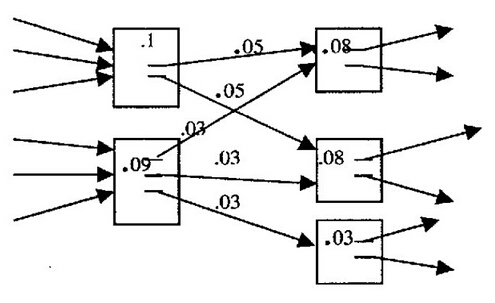
http://img.my.csdn.net/uploads/201209/20/1348120060_7086.png

继续假设B也有链接到C，并且D也有链接到包括A的3个页面。一个页面不能投票2次。所以B给每个页面半票。以同样的逻辑，D投出的票只有三分之一算到了A的PageRank上。

http://img.my.csdn.net/uploads/201209/20/1348120099_6078.png

换句话说，根据链出总数平分一个页面的PR值。

http://img.my.csdn.net/uploads/201209/20/1348120123_5802.png



由于存在一些出链为0，也就是那些不链接任何其他网页的网， 也称为孤立网页，使得很多网页能被访问到。因此需要对 PageRank公式进行修正，即在简单公式的基础上增加了**阻尼系数（damping factor）**q， q一般取值q=0.85。

其意义是，在任意时刻，用户到达某页面后并继续向后浏览的概率。 1- q= 0.15就是用户停止点击，随机跳到新URL的概率）的算法被用到了所有页面上，估算页面可能被上网者放入书签的概率。

最后，即所有这些被换算为一个百分比再乘上一个系数q。由于下面的算法，没有页面的PageRank会是0。所以，Google通过数学系统给了每个页面一个最小值。

http://img.my.csdn.net/uploads/201209/20/1348120862_4424.png

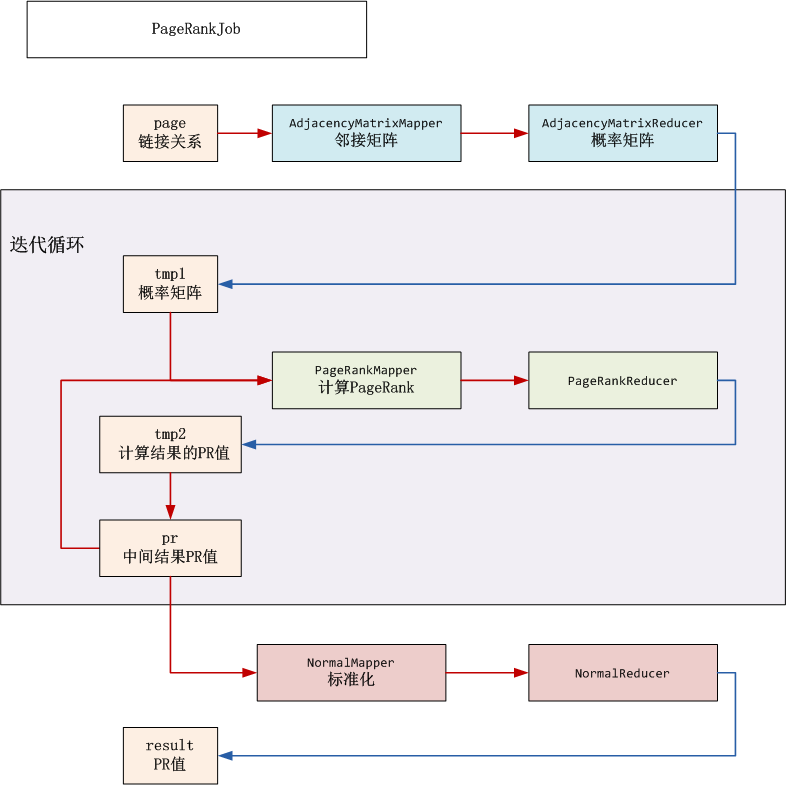
这个公式就是.S Brin 和 L. Page 在《The Anatomy of a Large- scale Hypertextual Web Search Engine Computer Networks and ISDN Systems 》定义的公式。

所以一个页面的PageRank是由其他页面的PageRank计算得到。Google不断的重复计算每个页面的PageRank。如果给每个页面一个随机PageRank值（非0），那么经过不断的重复计算，这些页面的PR值会趋向于正常和稳定。这就是搜索引擎使用它的原因。

**b. 你为什么选择使用这个系统/算法**

使用PageRank算法可以实现对数据集中的网页进行PR值的排名，从而可以得出数据集中的网页重要程度，即哪些网页被其他网页指向的比重高，可以实现题目的要求，并且最终可以实现大规模舆情分析的作用，创造社会价值。

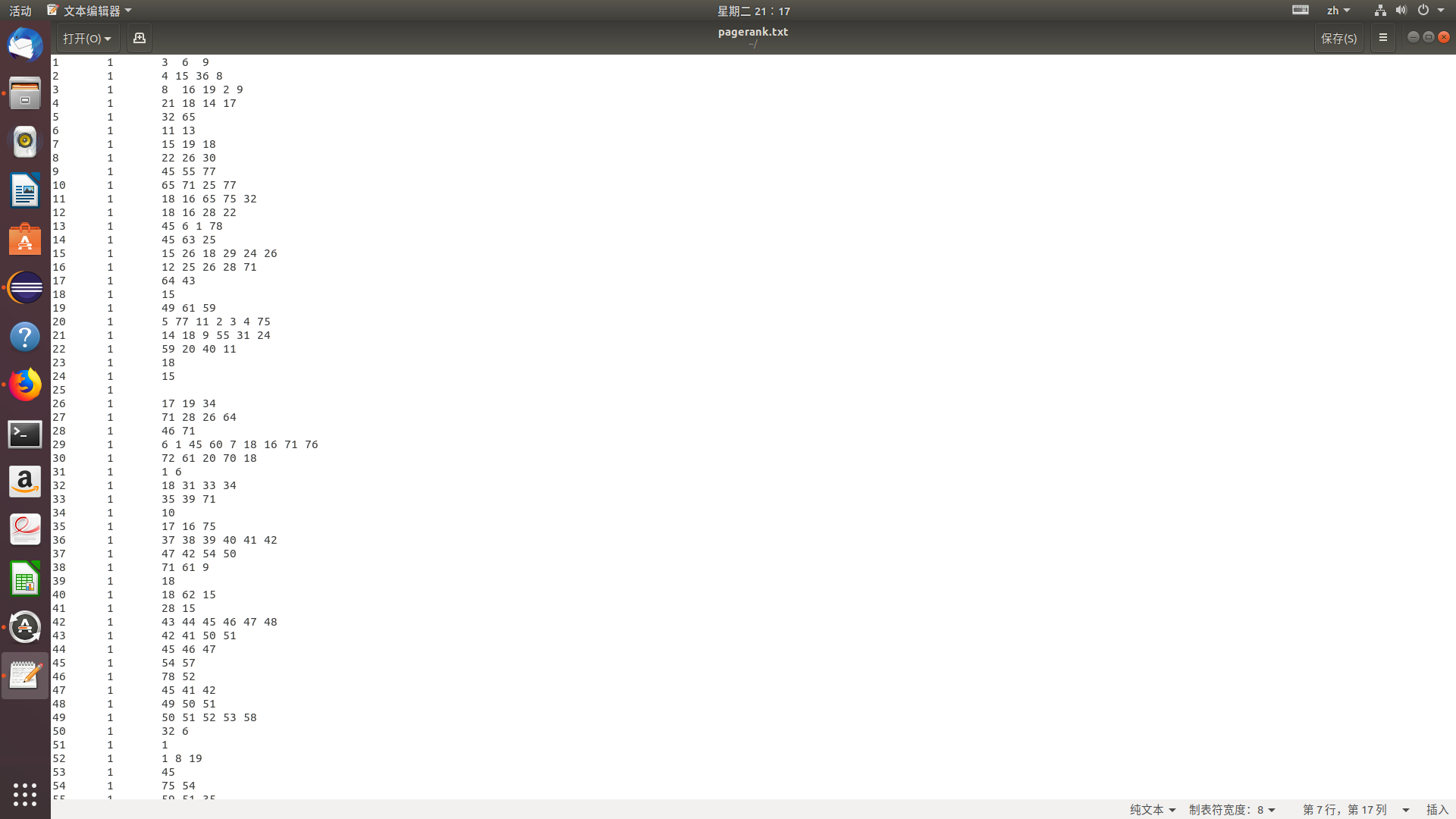
**c. 你是如何使用这个系统/算法的**



如上图所示是在MapReduce下的PageRank流程分解，首先提取出数据集中网页的URL值，**并且基于这些URL链接得出网页之间的互联关系（图或者邻接表的形式），我计划是使用Python的爬虫技术对大规模数据集中的所有URL进行索引，将每一个URL爬取出的网页内超链接与大规模数据集中的其余网页进行匹配，最终得出大规模数据集中的所有webURL之间的映射关系。**

**但是很显然，假设webURL的数量为n，那么这个爬虫系统的时间复杂度为O(n²)，这在数以万计、百万级的大规模URL数据下简直太可怕了，并且在我查阅的范围内并没有可以应用在大数据中的特大规模爬虫系统。**

**简而言之，我遇到了一个无法在大规模数据集下根据webURL得出网页PR值的难题。于是我自己生成了一些如下格式的数据集，并且将PageRank算法应用在自己的数据集上，也可以得出类似的实验结果。**

****

**数据集中的第一列代表网页的ID，第二列代表网页的默认初始PR值，这里我将其设置为1，第三列代表网页指向的其他网页的ID。**

**四. Https爬虫算法**

如PageRank算法中的介绍所述，我曾试图实现大规模webURL数据下的Https爬虫算法，但是最后因为复杂度的原因没有实现成功。

以下是Https爬虫算法的实现，使用Python语言，使用了urllib库以及正则表达式相关库。

# coding=utf-8  
import re  
import urllib.request  
import ssl  
  
  
def getHtml(url):  
 page = urllib.request.urlopen(url)  
 html = page.read()  
 html = html.decode('utf-8')  
 return html  
  
  
def getLink(html):  
 reg = r'<.\*?(href=".\*?").\*?'  
 img\_title = re.compile(reg)  
 imglist = re.findall(img\_title, html)  
 return imglist  
  
ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context  
url = "https://www.nytimes.com/2017/02/03/learning/tattoos.html"  
html = getHtml(url)  
linklist = getLink(html)  
  
print(linklist)

# 3. 系统设计/算法设计

介绍你设计实现的每一个系统/算法

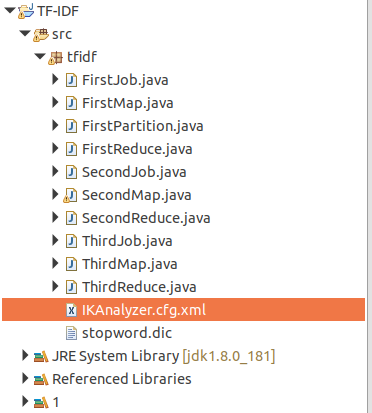
a. 系统架构/算法原理的描述，详细描述系统每个模块/算法每个步骤的功能和实现

b. 系统架构框图/算法流程图或伪代码

**一. TF-IDF算法**

**a. 系统架构/算法原理的描述，详细描述系统每个模块/算法每个步骤的功能和实现**

**Hadoop下实现TF-IDF算法的项目结构如下：**

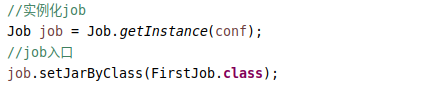


主要分为三个部分。

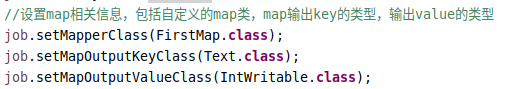
第一部分包括FirstJob.java; FirstMap.java; FirstPartition.java和FirstReduce.java四个文件。

**FirstJob.java主要实现了如下的功能：**

**实例化一个Job：**



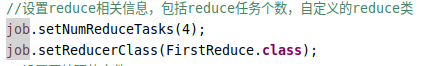
**设置Map相关信息，包括自定义的Map类以及Map输出key和value的类型：**



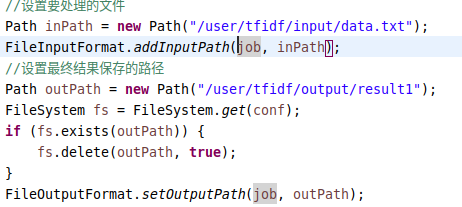
**设置Job的shuffle过程操作信息，包括分区、排序和分组：**

C:\Users\ASUS\AppData\Roaming\Tencent\Users\1031901796\TIM\WinTemp\RichOle\D8V76IU%7@CP59R`1)CBE(S.png

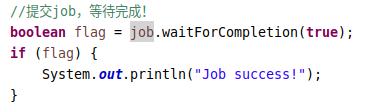
**设置Reduce相关信息，包括Reduce任务个数、自定义的Reduce类：**



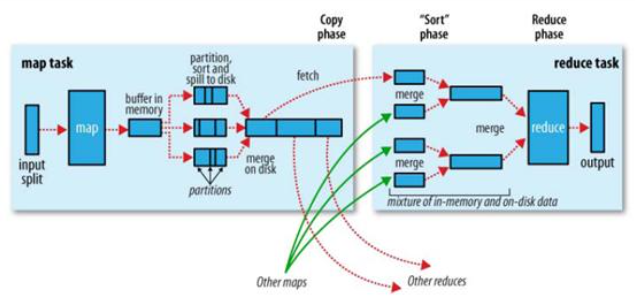
**设置要处理的文件和最终结果保存的路径：**



**最后提交Job，等待任务完成：**



Hadoop的核心思想是MapReduce，但shuffle又是MapReduce的核心。shuffle的主要工作是从Map结束到Reduce开始之间的过程。首先看下这张图，就能了解shuffle所处的位置。图中的partitions、copy phase、sort phase所代表的就是shuffle的不同阶段。



　shuffle阶段又可以分为Map端的shuffle和Reduce端的shuffle。

　　一、Map端的shuffle

　　Map端会处理输入数据并产生中间结果，这个中间结果会写到本地磁盘，而不是HDFS。每个Map的输出会先写到内存缓冲区中，当写入的数据达到设定的阈值时，系统将会启动一个线程将缓冲区的数据写到磁盘，这个过程叫做spill。

　　在spill写入之前，会先进行二次排序，首先根据数据所属的partition进行排序，然后每个partition中的数据再按key来排序。partition的目是将记录划分到不同的Reducer上去，以期望能够达到负载均衡，以后的Reducer就会根据partition来读取自己对应的数据。接着运行combiner(如果设置了的话)，combiner的本质也是一个Reducer，其目的是对将要写入到磁盘上的文件先进行一次处理，这样，写入到磁盘的数据量就会减少。最后将数据写到本地磁盘产生spill文件(spill文件保存在{mapred.local.dir}指定的目录中，Map任务结束后就会被删除)。

　　最后，每个Map任务可能产生多个spill文件，在每个Map任务完成前，会通过多路归并算法将这些spill文件归并成一个文件。至此，Map的shuffle过程就结束了。

　　二、Reduce端的shuffle

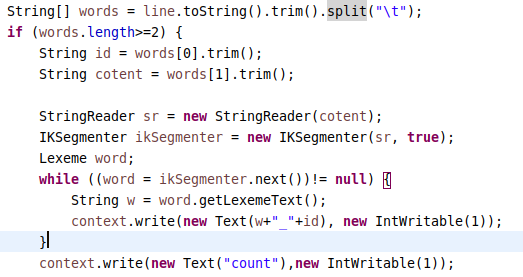
　　Reduce端的shuffle主要包括三个阶段，copy、sort(merge)和reduce。

　　首先要将Map端产生的输出文件拷贝到Reduce端，但每个Reducer如何知道自己应该处理哪些数据呢？因为Map端进行partition的时候，实际上就相当于指定了每个Reducer要处理的数据(partition就对应了Reducer)，所以Reducer在拷贝数据的时候只需拷贝与自己对应的partition中的数据即可。每个Reducer会处理一个或者多个partition，但需要先将自己对应的partition中的数据从每个Map的输出结果中拷贝过来。

　　接下来就是sort阶段，也成为merge阶段，因为这个阶段的主要工作是执行了归并排序。从Map端拷贝到Reduce端的数据都是有序的，所以很适合归并排序。最终在Reduce端生成一个较大的文件作为Reduce的输入。

最后就是Reduce过程了，在这个过程中产生了最终的输出结果，并将其写到HDFS上

**FirstMap.java主要实现了将数据进行分词并且以<词，1>和<count，1>的形式输出。在这里应用到了IKAnalyzer分词器:**



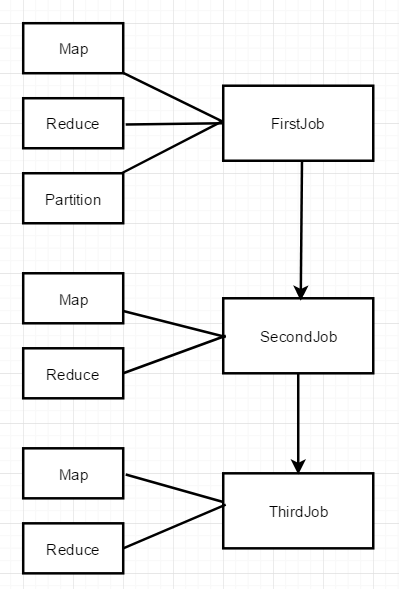
**FirstPartition.java实现了数据的分区处理。**

**FirstReduce.java即对流程1中的数据进行reduce处理。**

第二部分包括SecondJob.java; SecondMap.java和SecondReduce.java三个文件。主要的操作是对分区内的文件进行处理。

第三部分包括ThirdJob.java; ThirdMap.java和ThirdReduce.java三个文件。是算法的最后一步，对文件中的每个词进行TF-IDF的计算，并输出到文件系统中。

**b. 系统架构框图/算法流程图或伪代码**



TF-IDF算法执行的流程如上图所示

**二.PageRank算法**

**a. 系统架构/算法原理的描述，详细描述系统每个模块/算法每个步骤的功能和实现**

**1.PageRank原理分析：**

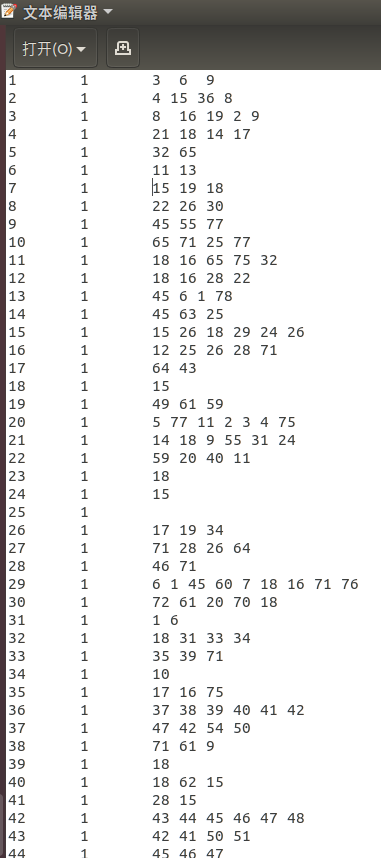
举例来讲：

假设每个网页都有一个自己的默认PR值，相当于人为添加给它是一种属性，用来标识网页的等级或者重要性，从而依据此标识达到排名目的。假设有ID号是1的一个网页，PR值是10，假如它产生了到ID=3，ID=6，ID=8，ID=9这4个网页的链接。那么可以理解为ID=1的网页向ID=3，6，8，9的4个网页各贡献了2.5的PR值。如果想求任意一个网页假设其ID=3的PR值，需要得到所有的其他网页对ID=3这个网页的贡献的总和，再按照函数“所求PR”=“总和”\*0.85+0.15得到。经过循环多次上述过程求得的网页PR值，可以作为网页排名的标识。

**2.MapReduce过程分析：**

**①准备数据：**

  理论数据是通过网页爬虫得到了某个特定封闭系统的所有网页的信息，但是对于本数据集来说，这是不可行的，具体原因在上文中已经说明。为了测试程序，我自己模拟生成自己定义特定格式的数据。下面是我用来测试的数据，存储方式如图：



（注：对于自定义模拟数据，在PR初始值的选取时，所有的网页是“平等”的，不会说自己写的网页和Google的热门网页有多少差别，但是按照某种法则经过一定运算后PR是不一样的，比如很多其他的网页可能会链接到google，它的PR自然会比你的高。所以初始值的选取按照这种逻辑来讲符合现实些，即所有网页默认PR值相等。但是即使初始值定的不一样 ，整个系统的PR总和可能会变化，最后的每个网页PR也会发生变化，但是这种量之间的变化，不会影响到网页自身的通过比较大小方式上的逻辑排名。）

**②Map过程：**

map接受的数据格式默认是<偏移量，文本行>这样的<key,value>对，形如<0,1    5  2 3 4 5><20,2    10 3 5 8 9>。

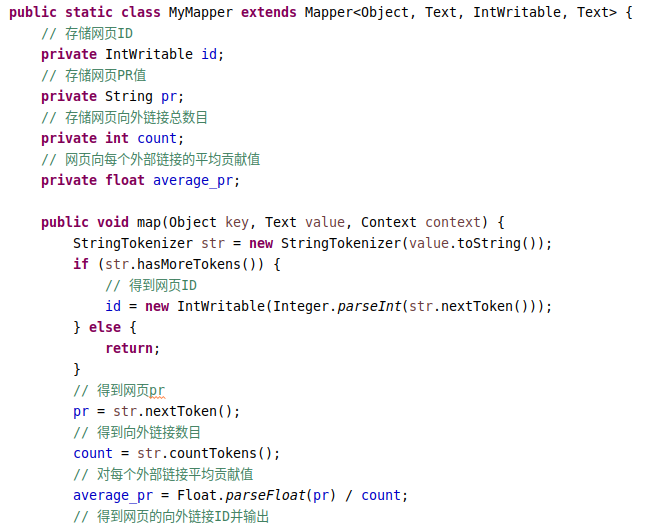
**目标：**将默认数据格式，转换成自定义格式<key,value>对。

**已知：**Hadoop框架在Map阶段的时候会自动实现sort过程，就是将相同的key的所有value保存到list，形如<key,list(1,1,1,2)>这种形式，例如上述对ID=2的网页有ID=1，6，7，8.这4个网页贡献（1.25，1，5/3，5），那么如果你采用的key是网页ID，那么Hadoop框架会以此种形式<2,list(1.25，1，5/3，5)>输出。

**分析：**因为这个过程要进行多次，reduce的最终输出结果需要保存成原文件那样的格式，所以每个网页ID和自己链接情况也要在map阶段输出给reduce。

**操作：**所以对于上述第一行操作map函数后结果是<id=1,2><id=1,3><id=1,4>,<id=1,5>保存了id=1网页的链接情况，同时还要输出<id=2,1.25><id=3,1.25><id=4,1.25><id=5,1.25>,每个网页得到的贡献值。

**代码：**

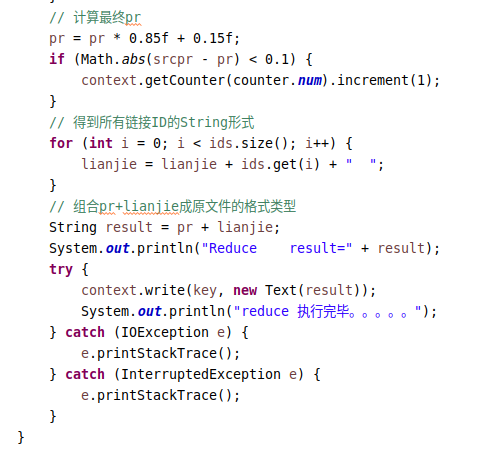


**3.Reduce过程分析：**

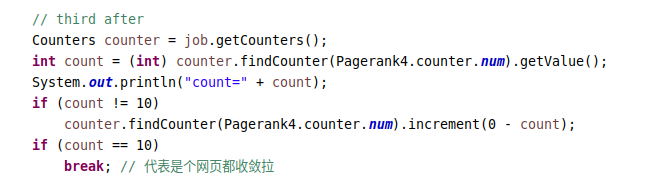
**分析：**这个阶段操作就相对简单很多，读取map的输出<key,value>，并解析出来。

**具体操作：**如果value中首字母是“@”表示是贡献值，如果是“&”表示是链接情况。

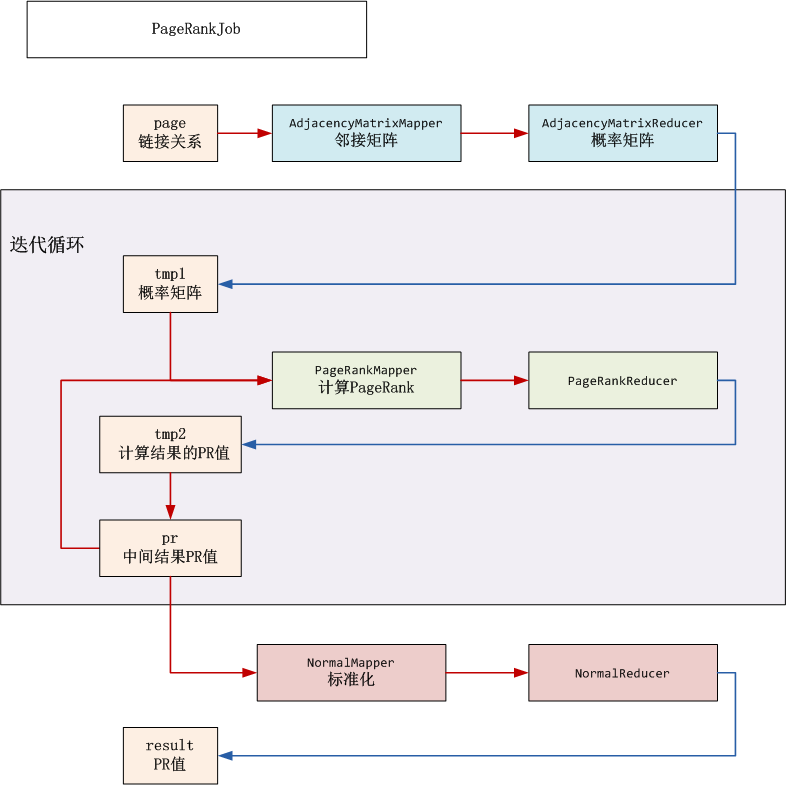
**代码：**



**4.main函数：**



**b. 系统架构框图/算法流程图或伪代码**



Hadoop下实现PageRank的系统架构图如上图所示。

# 4. 实验流程

详细介绍你完成实验的主要流程

a. 主要过程描述，如系统搭建，论文阅读，算法设计实现等

b. 每个算法/系统模块输入数据示例截图，并作详细描述

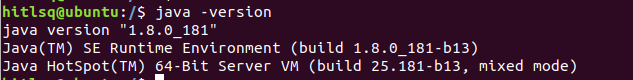
c. 每个算法/系统模块输出结果示例截图，并作详细描述

**1.配置实验环境**

在Linux下配置hadoop会比在windows系统下配置hadoop减少很多不必要的麻烦，所以配置实验环境的第一步是配置ubuntu虚拟机。实验使用的是VMware WorkStation 15 pro和Ubuntu 18.04版本。

配置好linux系统后，首先需要安装jdk。安装的方法谷歌上有很多，就不在这里详细叙述了。

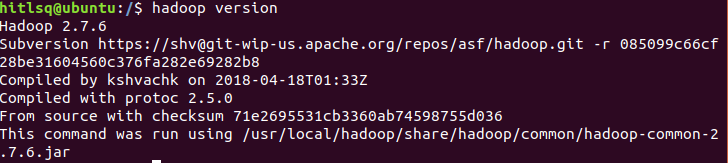
配置好jdk之后在终端输入java –version，即可查看jdk的版本：



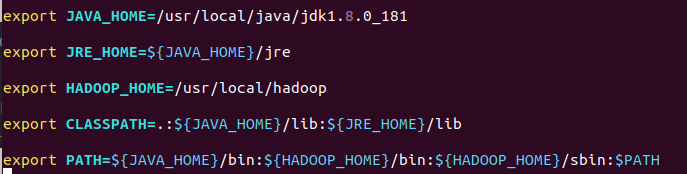
安装hadoop的教程谷歌上也有很多，在这里就不详细叙述了。

**【踩坑经历：不要安装hadoop2.9.1版本，因为此版本的hadoop在eclipse中配置的时候会出现缺失配置文件以及配置之后无法点击Add new location等错误。我最开始安装的是hadoop2.9.1版本，但是卡在了eclipse配置这里，最后只能推倒重来，最终安装的是hadoop2.7.6版本】**

在终端输入hadoop version即可查看hadoop的版本：（如果想在任何位置都能通过终端控制hadoop，那么需要将hadoop配置环境变量）

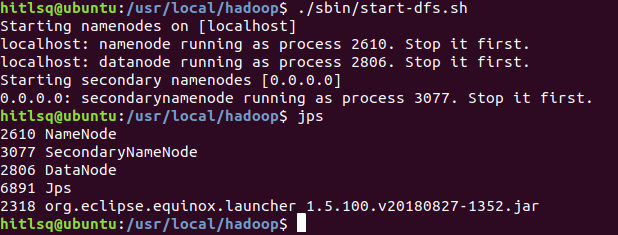


配置环境变量的方法：在终端输入vim /etc/bash.bashrc，然后在文件的末尾增加如下五行即可。（需要根据自己的配置路径来添加）



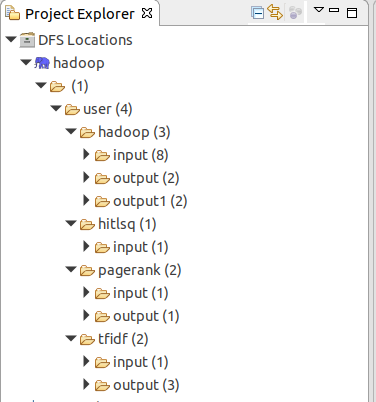
在eclipse for java ee下配置hadoop的方法主要参考网上的博客就可以，**【踩坑经历：需要注意的是路径的设置以及根据不同的hadoop的配置文件更改build.xml中的配置项。Build.xml中的配置项必须和hadoop目录下的jar包的名字相同】**

全部配置完成之后，启动hadoop，然后在终端输入jps查看hadoop的部署情况：



在终端输入jps后，如果出现NameNode，SecondaryNameNode，DataNode，Jps四项，则表明hadoop配置成功。出现最后一项则表示hadoop已经成功部署在了eclipse中。

配置完成之后，在eclispe中就可以查看文件系统中的文件了，如下图所示：



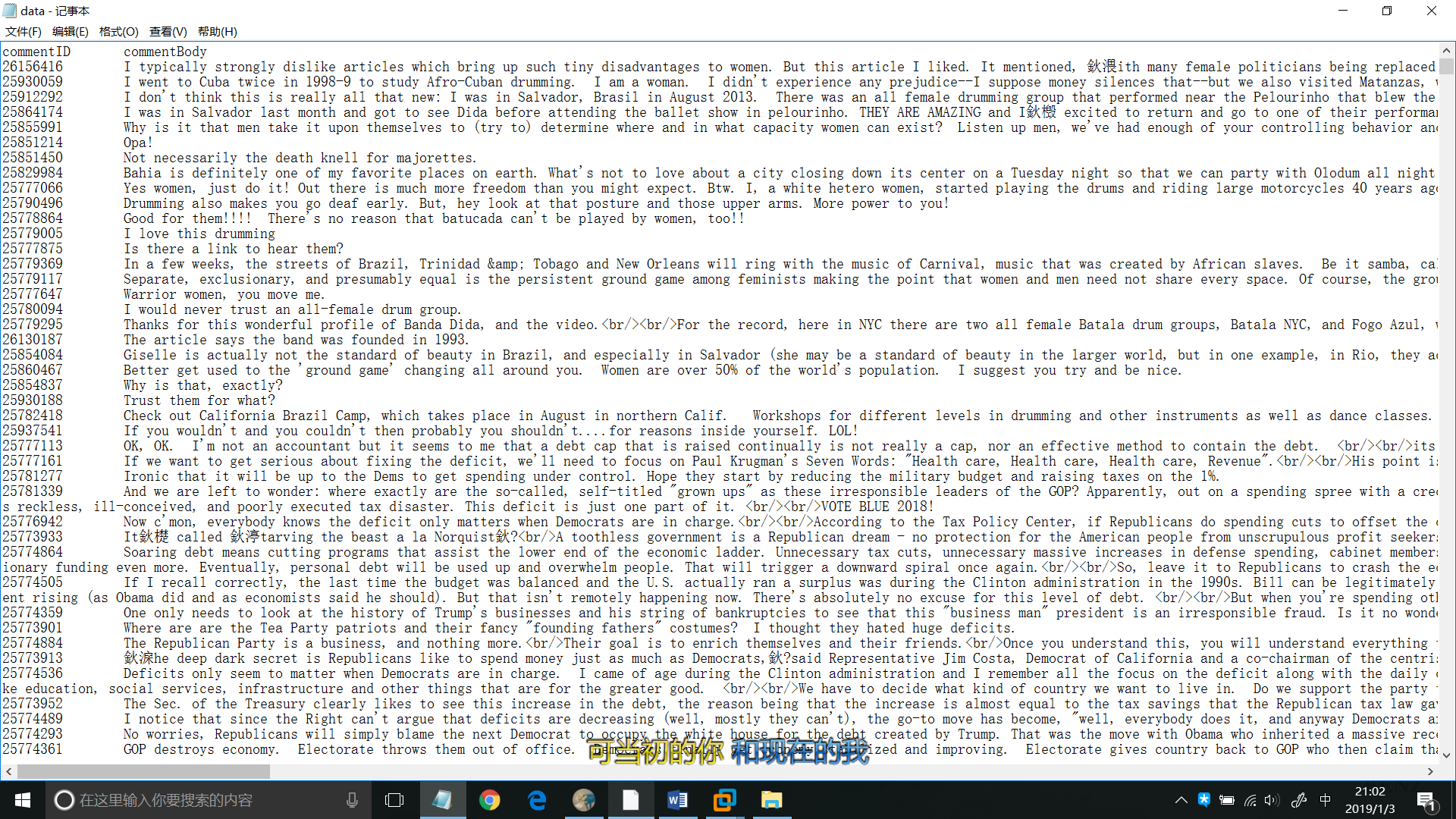
**主要参考：网上的技术博客；《hadoop实战》书籍**

**【踩坑经历：在问题1（TF-IDF）算法中应用到了java IKAnalyzer分词器，使用该分词器时必须在项目中导入分词词典以及相关的jar包。】**

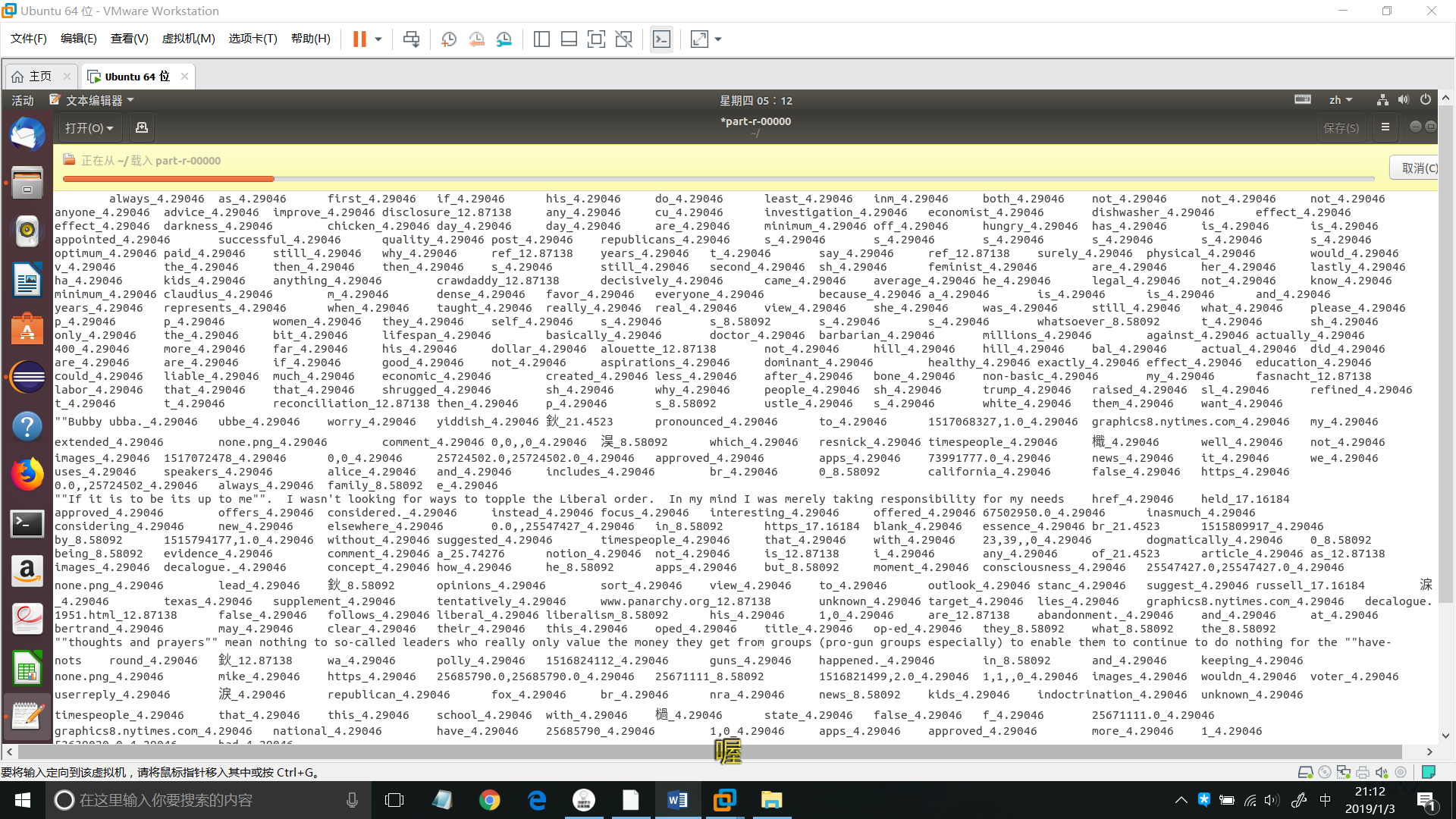
**【踩坑经历：最好不要在文件系统中存放过大的文件，有可能超出限制而导致hadoop进入安全模式。】**

**2.TF-IDF算法输入数据、输出数据**

输入数据是从所给链接下载的comments评论集中的评论和评论编号，单个文件总共22万行左右，部分数据的截图如下：

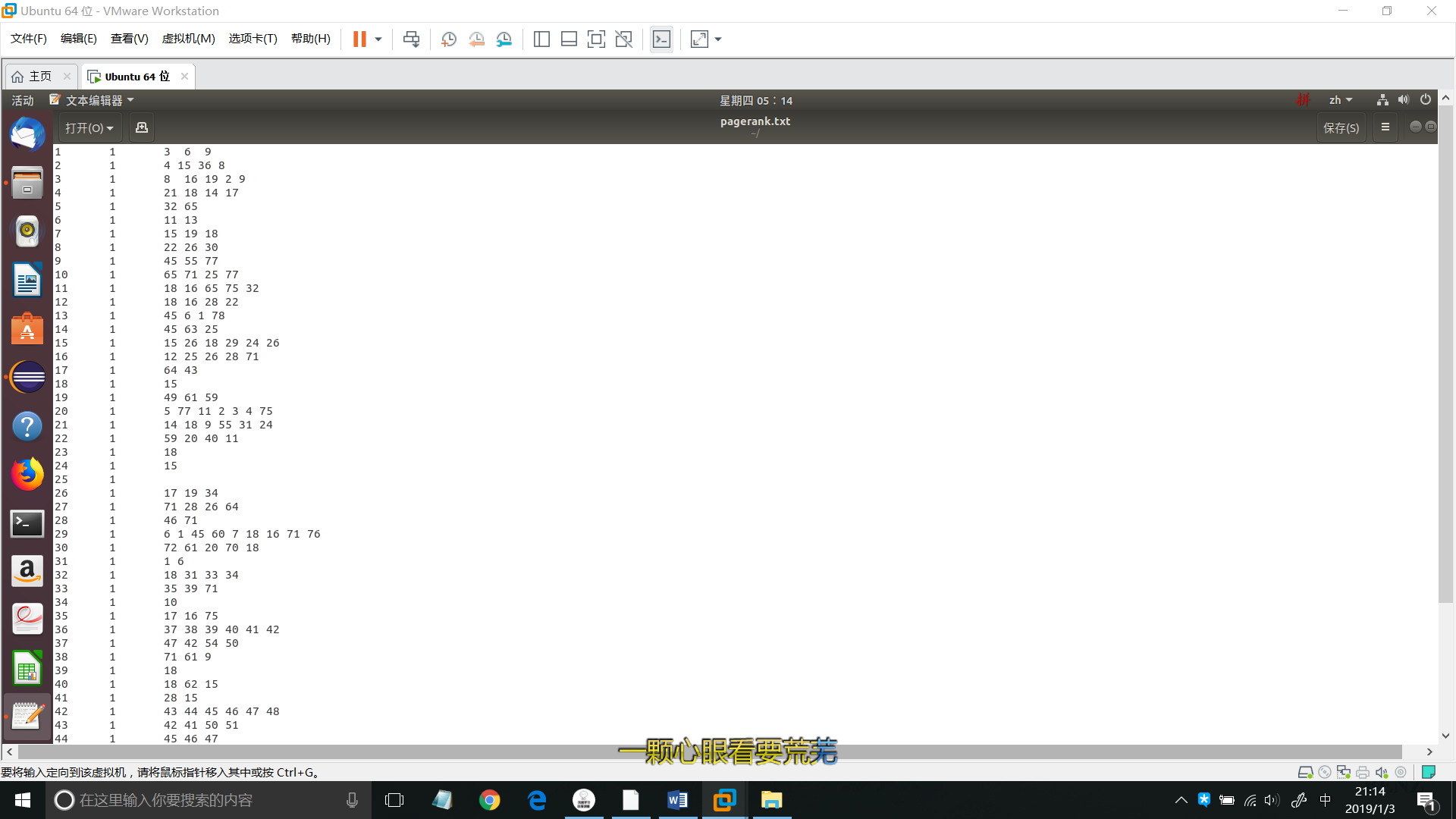


输出数据是评论集中的每个词的TF-IDF值，用户可以输入关键字/词进行TF-IDF值的检索。

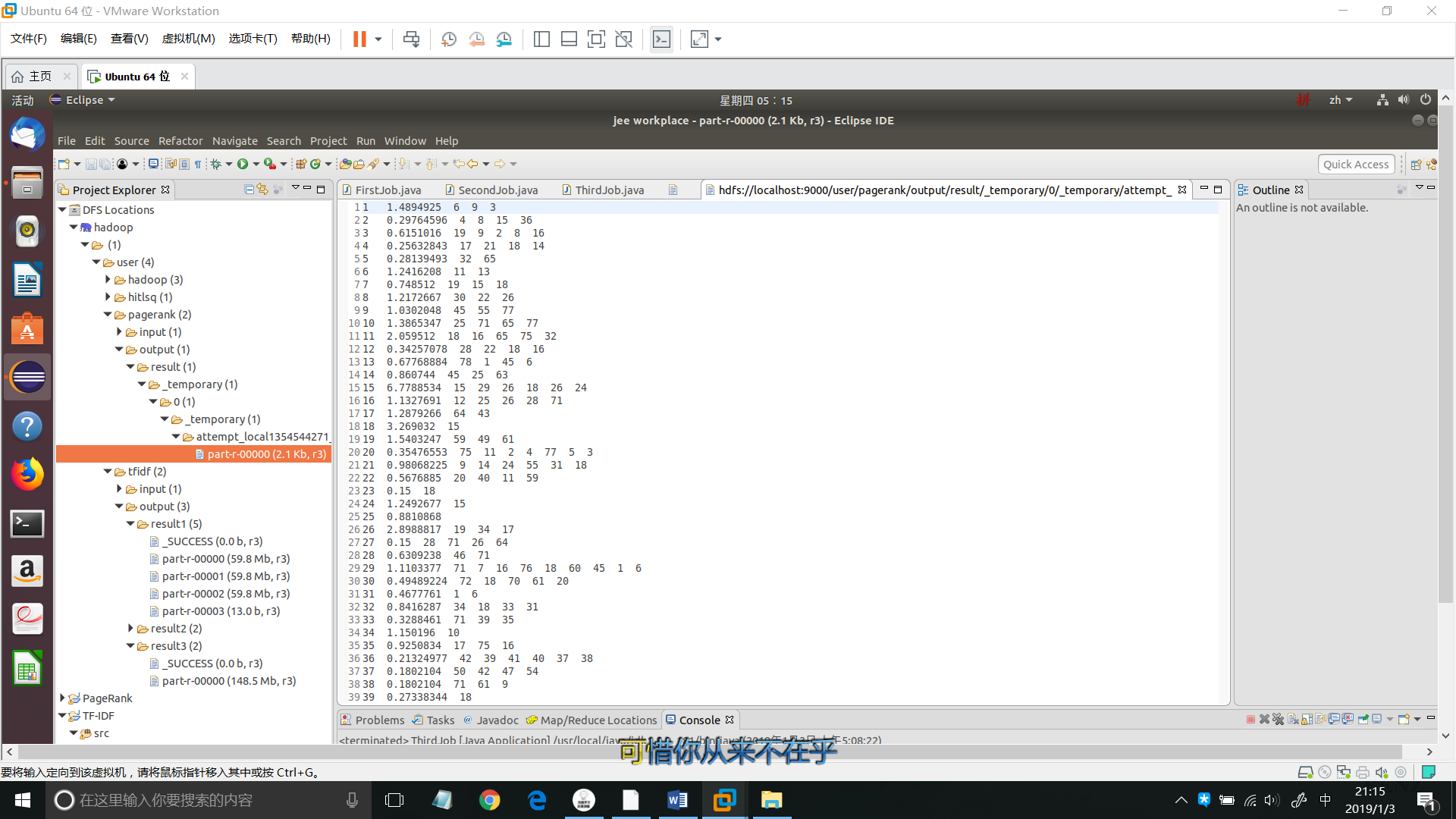


**3. PageRank算法输入数据、输出数据**

PageRank算法的输入数据是url的id，初始pr值，关联url的id，数据是自己生成的，原因在上文中已经进行了解释，数据的截图如下：



输出的数据是每一个网页的pr值，用户下载文件之后可以提取出pr值并对其进行排序。



# 5. 实验

测试你设计实现的系统/算法的效果

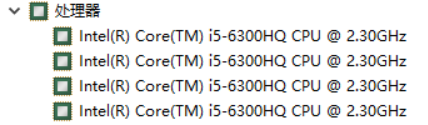
a. 测试环境：CPU、内存、磁盘、网络（如有）、GPU（如有）

b. 数据集：数据集的来源（是自己生成的还是下载的，生成程序是如何编写的，下载的数据是选取了一部分还是全部使用了）、数据文件或数据库占用磁盘空间大小、包含的记录个数（行数）、包含的属性及其意义

c. 测试算法/系统的效果，如误差大小，准确率等随参数的变化情况，用图表展示并作详细描述

d. 测试算法/系统的性能，如运行时间，中间结果大小，数据在不同进程间传输代价等随输入规模变化的情况，用图表展示并作详细描述

**1. 测试环境：**

**CPU：**

**内存：4G**

**磁盘：50G**

**网络：HIT-WLAN**

**GPU：无**

**2. 数据集：**

**数据集的来源：第一题：下载的**

**第二题：自己生成的**

**第一题下载的数据全部使用了，数据文件占据156Mb左右磁盘空间，行数多于220000行（即多于220000条数据），每一行数据对应着一个评论的Id和内容。**

**第二题的生成程序是python随机数程序，生成了80000组数据，提取出其中80组数据作为测试，因为是随机数程序，而且总数据量以及计算复杂度都不大，所以数据量对程序的影响并不大。每一行数据对应一个webURL的id、pr值和与其相关的webURL的id。**

**3. 测试算法/系统的效果**

**①第一题——TF-IDF算法**

分别取1000行、10000行、100000行、200000行以及全部数据进行实验，实验结果表明，TF-IDF算法的准确率和误差大小与数据量并未呈现出较强的相关性。

**①第二题——PageRank算法**

随着webURL数量的增加，webURL的pr值的准确性和误差大小都没有明显变化。但是，当单个url所指向的其他url的数目增多时，准确性和误差大小都有了比较明显的下降。

**4. 测试算法/系统的性能**

此部分给出了基于实际数据的TF-IDF算法的性能分析，基于随机数据的PageRank算法可能会因为随机性的原因无法体现出数据对算法的影响。

**①第一题——TF-IDF算法**

如表所示

FirstJob.java

|  |  |
| --- | --- |
| 数据行数 | 运行时间 |
| 1000 | 3s |
| 10000 | 14s |
| 100000 | 56s |
| 全部数据 | 1min55s |

|  |  |
| --- | --- |
| 数据大小 | 中间结果大小 |
| 1000 | <1Mb |
| 10000 | 9Mb |
| 100000 | 90.5Mb |
| 全部数据 | 179.4Mb |

SecondJob.java

|  |  |
| --- | --- |
| 数据行数 | 运行时间 |
| 1000 | 很快 |
| 10000 | 6s |
| 100000 | 22s |
| 全部数据 | 38s |

|  |  |
| --- | --- |
| 数据行数 | 中间结果大小 |
| 1000 | <<1Mb |
| 10000 | <1Mb |
| 100000 | 1.3Mb |
| 全部数据 | 2.4Mb |

ThirdJob.java

|  |  |
| --- | --- |
| 数据行数 | 运行时间 |
| 1000 | 很快 |
| 10000 | 11s |
| 100000 | 38s |
| 全部数据 | 1min11s |

|  |  |
| --- | --- |
| 数据行数 | 中间结果大小 |
| 1000 | 很小 |
| 10000 | 11Mb |
| 100000 | 76.8Mb |
| 全部数据 | 148.5Mb |

# 6. 结论

介绍你实验的结论，如某个算法效果如何，某种方案是否可行等。

实验结论：TF-IDF算法在数据量很大的时候运行时间会很长，不适用于大规模的数据分析，PageRank算法在处理已经被整理好的数据的时候可以发挥出很好的效果，但是大多数实际的数据无法直接应用到PageRank算法当中，因为从一个大规模封闭数据集中得到每一个网页的webURL之间的相关性是很困难的事情。

**报告说明：**

**1、不强制要求使用该模版写报告，该模版的样式仅供参考。已经开始撰写报告的同学，或想使用Latex撰写报告的同学请自由发挥；**

**2、请尽量在给定的框架中进行补充，覆盖到给定的点；可以在框架的基础上弹性修改，但是主要内容需要有；请尽可能写清楚你在实验过程（尤其是搭配环境）中的踩坑细节，越详细越好；**

**3、报告字数要求在2000字以上。**