# 華中科技大学

# 课 程 实 验 报 告

**课程名称：** 大数据分析

专业班级： **CS2011**

学　　号： **U202014774**

姓　　名： **王逸**

指导教师： **崔金华**

报告日期： **2022.12.28**

计算机科学与技术学院

# 目录

# 实验一 wordCount 算法及其实现

## 1.1 实验目的

1、理解 map-reduce 算法思想与流程；
2、应用 map-reduce 思想解决 wordCount 问题；
3、（可选）掌握并应用 combine 与 shuffle 过程。

## 1.2 实验内容

提供 9 个预处理过的源文件（source01-09）模拟 9 个分布式节点，每个源文件中包含一百万个由英文、数字和字符（不包括逗号）构成的单词，单词由逗号与换行符分割。

要求应用 map-reduce 思想，模拟 9 个 map 节点与 3 个 reduce 节点实现 wordCount 功能，输出对应的 map 文件和最终的 reduce 结果文件。由于源文件较大，要求使用多线程来模拟分布式节点。

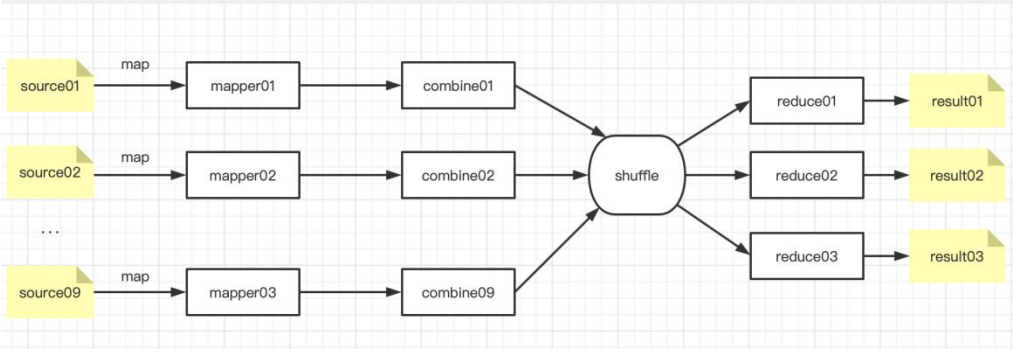
学有余力的同学可以在 map-reduce 的基础上添加 combine 与 shuffle 过程，并可以计算线程运行时间来考察这些过程对算法整体的影响。

提示：实现 shuffle 过程时应保证每个 reduce 节点的工作量尽量相当，来减少整体运行时间。

## 1.3 实验过程

### 1.3.1 编程思路

该实验总体流程图如下所示：主要分为 map，combine 和 shufle 三个部分。



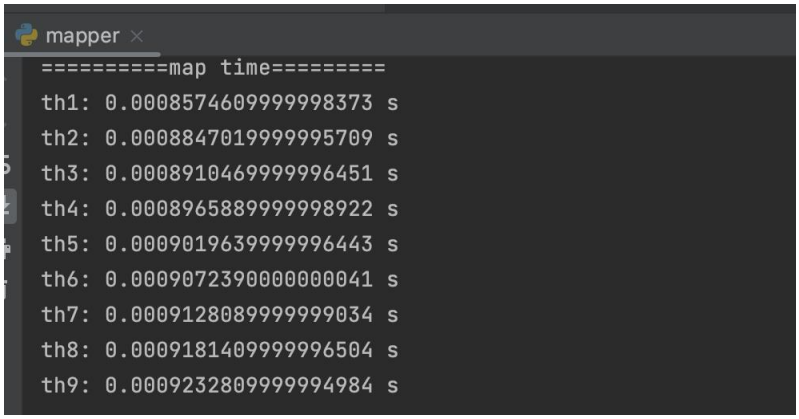
1) Map——mapper.py 文件中包含两个函数：
   a. read_line（source_file）：传入参数为被读取的文件，对于文件里的每一行使用 strip()函数去掉行末尾的回车符，再调用 split(‘, ‘)函数，返回一个行单词的迭代器。
   b. Mapper(read_file, write_file)：传入参数为被读取的文件路径，以及写入文件路径。该部分被读取文件为 source0x，写入文件为 mapper0x。使用 open 函数打开 read_file, 调用 read_line()函数, 将每行单词以(word, 1)的形式写入 mapper0x 文件中。
   c. 主函数：通过调用 threading.Tread()函数，用 9 个线程并行处理对应的 source 文件并写入到 mapper 文件中，最后输出每个线程的处理时间。
2) Combine——combine.py 文件中定义了一个函数：
   a. combine(read_file, write_file)：传入参数为被读文件路径和写入文件路径。采用字典存储单词对应的出现次数。被读取的文件中每一行是以 word，1 形式存储，调用 strip 和 split 函数获得单词 word，如果该单词已经存在于字典的键值中，则对应数目加 1，否则 dict[word]=1. 文件遍历结束后, 调用 sorted 函数将字典按键值大小排序, 最后将(key, value）写入文件中。
   b. 主函数：与 mapper 中类似，9 个线程并行处理 mapper 文件并写入到对应的 combine 文件中，最后输出每个线程的处理时间。
3) shuffle——shuffle.py 文件中定义了一个函数：
   a. shuffle(readfile): 将文件中的单词按照首字母分到三个 shuffle 文件中。由于是将 9 个 combine 文件中的单词分到固定的三个 shuffle 文件中，所以写入文件的打开方式应该为’a’。
4) Reduce——reduce.py 中定义了 reduce 函数：
   a. Reduce(readfile, writefile)：处理方式与 combine 相同，这里不过多赘述。
   b. 主函数: 使用三个线程处理对应的 shuffle 文件并写入到相应的 reduce 文件中。

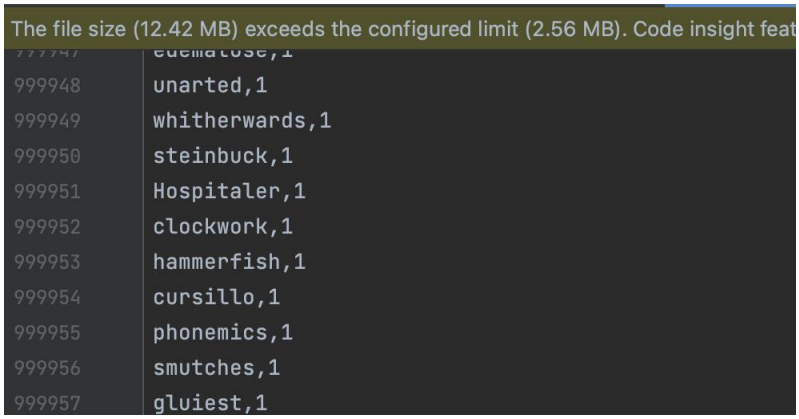
### 1.3.2 遇到的问题及解决方式

map-reduce 的流程在 ppt 与课堂上里已经讲解的很详细了，按照课上介绍的步骤一步一步来没有太大问题。由于之前接触过 python，所以在代码编写上比较顺利，不过在这次实验中接触到了 yield 函数以及线程函数的使用方法，学习到了新的知识。

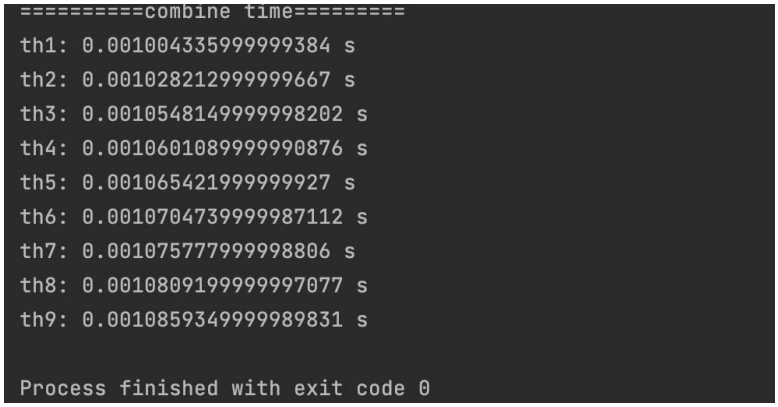
### 1.3.3 实验测试与结果分析

Mapper：处理时间与文件部分内容




combine：处理时间与文件部分内容




shuffle：处理时间与文件部分内容

```
==========shuffle time=========
th1: 0.0012074169999998219 s
th2: 0.0012653729999998475 s
th3: 0.0012713719999997153 s
th4: 0.0012998029999993221 s
th5: 0.0013050149999997984 s
th6: 0.00131028699999991 s
th7: 0.0013154039999996314 s
th8: 0.0013206429999996772 s
th9: 0.0013258715000000116 s

Process finished with exit code 0
```

```
A-axes,3
A-blast,1
A-bomb,2
A-day,3
A-flat,1
A-line,1
A-one,3
A-scope,1
A-shaped,2
A-sharp,1
A-tent,4
```

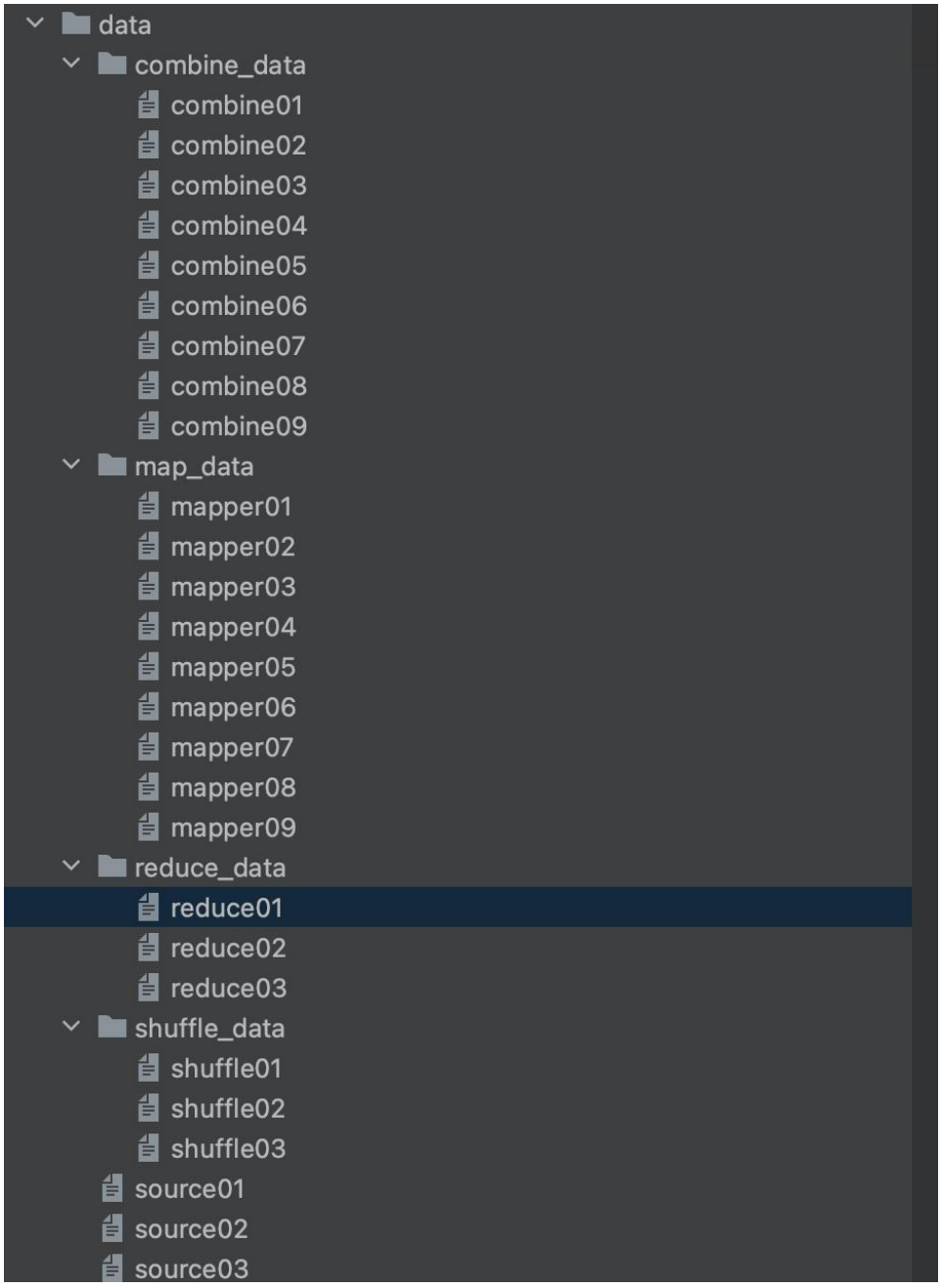
reduce：处理时间与文件部分内容

```
==========reduce time=========
th1: 0.00027910699999988964 s
th2: 0.000301776999999781 s
th3: 0.0003081849999997388 s

Process finished with exit code 0
```

```
A-P,22
A'asia,36
A-1,36
A-OK,32
A-and-R,38
A-axes,42
A-axis,34
```

**输出文件与源文件的结构如下：**




## 1.4 实验总结

通过对 map-reduce 的实现，更加深入了理解了该算法的处理流程以及实现方式，其中通过添加 combine 和 shuffle 部分，减少了线程的运行时间。在本次实验过程中对文件读取写入以及打开方式有了进一步认识，更加熟练地使用字典类型来解决单词对应频率的计算。