

**数学与信息学院学生实验报告**

**实验课程名称：**大数据技术概论 **教师： \_\_**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **实验五：MapReduce初级编程实践** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** |  | **年级专业班级** |  |
| **小组成员** | **无** | | | **实验日期** |  |

# 1. 实验目的和要求

## 1.1 实验目的

① 通过实验掌握基本的MapReduce编程方法；

② 掌握用MapReduce解决一些常见的数据处理问题，包括数据去重、数据排序和数据挖掘等。

## 1.2实验软硬件环境

① 操作系统：Linux（Ubuntu16.04）；

② Hadoop版本：3.2.0。

## 1.3 实验要求

**（一）编程实现文件合并和去重操作**

对于两个输入文件，即文件A和文件B，请编写MapReduce程序，对两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件C。下面是输入文件和输出文件的一个样例供参考。

输入文件A的样例如下：

|  |
| --- |
| 20170101 x  20170102 y  20170103 x  20170104 y  20170105 z  20170106 x |

输入文件B的样例如下：

|  |
| --- |
| 20170101 y  20170102 y  20170103 x  20170104 z  20170105 y |

根据输入文件A和B合并得到的输出文件C的样例如下：

|  |
| --- |
| 20170101 x  20170101 y  20170102 y  20170103 x  20170104 y  20170104 z  20170105 y  20170105 z  20170106 x |

**（二）编写程序实现对输入文件的排序**

现在有多个输入文件，每个文件中的每行内容均为一个整数。要求读取所有文件中的整数，进行升序排序后，输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数。下面是输入文件和输出文件的一个样例供参考。

输入文件1的样例如下：

|  |
| --- |
| 33  37  12  40 |

输入文件2的样例如下：

|  |
| --- |
| 4  16  39  5 |

输入文件3的样例如下：

|  |
| --- |
| 1  45  25 |

根据输入文件1、2和3得到的输出文件如下：

|  |
| --- |
| 1 1  2 4  3 5  4 12  5 16  6 25  7 33  8 37  9 39  10 40  11 45 |

**（三）对给定的表格进行信息挖掘**

下面给出一个child-parent的表格，要求挖掘其中的父子辈关系，给出祖孙辈关系的表格。

输入文件内容如下：

|  |
| --- |
| child parent  Steven Lucy  Steven Jack  Jone Lucy  Jone Jack  Lucy Mary  Lucy Frank  Jack Alice  Jack Jesse  David Alice  David Jesse  Philip David  Philip Alma  Mark David  Mark Alma |

输出文件内容如下：

|  |
| --- |
| grandchild grandparent  Steven Alice  Steven Jesse  Jone Alice  Jone Jesse  Steven Mary  Steven Frank  Jone Mary  Jone Frank  Philip Alice  Philip Jesse  Mark Alice  Mark Jesse |

# 2. 实验记录

**（一）编程实现文件合并和去重操作**

对于两个输入文件，即文件A和文件B，请编写MapReduce程序，对两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件C。下面是输入文件和输出文件的一个样例供参考。

输入文件A的样例如下：

|  |
| --- |
| 20170101 x  20170102 y  20170103 x  20170104 y  20170105 z  20170106 x |

输入文件B的样例如下：

|  |
| --- |
| 20170101 y  20170102 y  20170103 x  20170104 z  20170105 y |

根据输入文件A和B合并得到的输出文件C的样例如下：

|  |
| --- |
| 20170101 x  20170101 y  20170102 y  20170103 x  20170104 y  20170104 z  20170105 y  20170105 z  20170106 x |

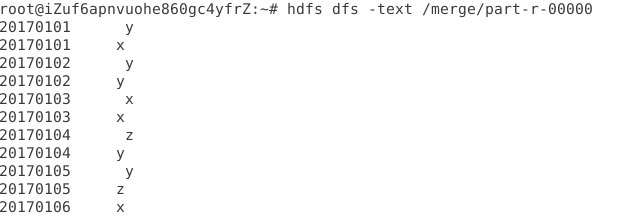
**代码：**

1. **import** org.apache.hadoop.hbase.util.Merge;
2. **import** org.apache.hadoop.io.Text;
3. **import** java.io.IOException;
4. **import** org.apache.hadoop.conf.Configuration;
5. **import** org.apache.hadoop.fs.Path;
6. **import** org.apache.hadoop.hbase.mapreduce.IndexBuilder.Map;
7. **import** org.apache.hadoop.mapreduce.Job;
8. **import** org.apache.hadoop.mapreduce.Mapper;
9. **import** org.apache.hadoop.mapreduce.Reducer;
10. **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11. **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12. **import** org.joni.constants.Reduce;
14. **public** **class** Main {
15. **public** **static** **void** main(String[] args) **throws** Exception {
16. Configuration conf = **new** Configuration();
17. conf.set("fs.defaultFS", "hdfs://localhost:9000");
18. Job job = Job.getInstance(conf, "合并除重");
19. job.setJarByClass(Merge.**class**);
20. job.setMapperClass(Map.**class**);
21. job.setReducerClass(Reduce.**class**);
22. job.setOutputKeyClass(Text.**class**);
23. job.setOutputValueClass(Text.**class**);
24. FileInputFormat.addInputPath(job, **new** Path("/input"));
25. FileOutputFormat.setOutputPath(job, **new** Path("/merge"));
26. System.exit(job.waitForCompletion(**true**) ? 0 : 1);
27. }
28. **public** **static** **class** Map **extends** Mapper<Object, Text, Text, Text> {
29. **private** **static** Text text = **new** Text();
31. @Override
32. **public** **void** map(Object key, Text val, Context con) **throws** IOException, InterruptedException {
33. text = val;
34. con.write(text, **new** Text(""));
35. }
36. }
38. **public** **static** **class** Reduce **extends** Reducer<Text, Text, Text, Text> {
39. @Override
40. **public** **void** reduce(Text key, Iterable<Text> val, Context con) **throws** IOException, InterruptedException {
41. con.write(key, **new** Text(""));
42. }
43. }
44. }

先将A和B上传到指定文件夹下：



运行代码后查看合并文件：



对比样例答案，符合预期（因为直接复制样例，而样例所给两个文件的空格数不一样，所以我这里合并后会出现空格不对齐的情况，所以虽然和样例输出的格式不一样但我也是对的）。

**（二）编写程序实现对输入文件的排序**

现在有多个输入文件，每个文件中的每行内容均为一个整数。要求读取所有文件中的整数，进行升序排序后，输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数。下面是输入文件和输出文件的一个样例供参考。

输入文件1的样例如下：

|  |
| --- |
| 33  37  12  40 |

输入文件2的样例如下：

|  |
| --- |
| 4  16  39  5 |

输入文件3的样例如下：

|  |
| --- |
| 1  45  25 |

根据输入文件1、2和3得到的输出文件如下：

|  |
| --- |
| 1 1  2 4  3 5  4 12  5 16  6 25  7 33  8 37  9 39  10 40  11 45 |

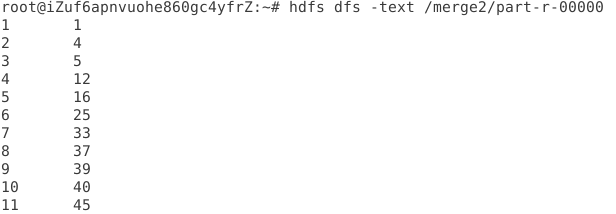
**代码：**

1. **import** java.io.IOException;
2. **import** org.apache.hadoop.conf.Configuration;
3. **import** org.apache.hadoop.fs.Path;
4. **import** org.apache.hadoop.io.\*;
5. **import** org.apache.hadoop.mapreduce.\*;
6. **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
7. **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
8. **import** org.apache.hadoop.util.GenericOptionsParser;
10. **public** **class** Main {
11. **public** **static** **void** main(String[] args) **throws** Exception {
12. Configuration conf = **new** Configuration();
13. conf.set("fs.defaultFS", "hdfs://localhost:9000");
14. Job job = Job.getInstance(conf, "合并排序");
15. job.setMapperClass(Map.**class**);
16. job.setReducerClass(Reduce.**class**);
17. job.setPartitionerClass(Partition.**class**);
18. job.setOutputKeyClass(IntWritable.**class**);
19. job.setOutputValueClass(IntWritable.**class**);
20. FileInputFormat.addInputPath(job, **new** Path("/input"));
21. FileOutputFormat.setOutputPath(job, **new** Path("/merge2"));
22. System.exit(job.waitForCompletion(**true**) ? 0 : 1);
23. System.exit(job.waitForCompletion(**true**) ? 0 : 1);
24. }
25. **public** **static** **class** Map **extends** Mapper<Object, Text, IntWritable, IntWritable> {
26. **private** **static** IntWritable data = **new** IntWritable();
28. @Override
29. **public** **void** map(Object key, Text val, Context con) **throws** IOException, InterruptedException {
30. String line = val.toString();
31. data.set(Integer.parseInt(line));
32. con.write(data, **new** IntWritable(1));
33. }
34. }
36. **public** **static** **class** Reduce **extends** Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {
37. **private** **static** IntWritable num = **new** IntWritable(1);
39. @Override
40. **public** **void** reduce(IntWritable key, Iterable<IntWritable> val, Context con) **throws** IOException, InterruptedException {
41. **for** (IntWritable v : val) {
42. con.write(num, key);
43. num = **new** IntWritable(num.get() + 1);
44. }
45. }
46. }
47. **public** **static** **class** Partition **extends** Partitioner<IntWritable, IntWritable> {
48. **public** **int** getPartition(IntWritable key, IntWritable val, **int** numPartitions){
49. **int** Maxnumber = 65223;
50. **int** bound = Maxnumber / numPartitions + 1;
51. **int** Keynumber = key.get();
52. **for**(**int** i = 0; i < numPartitions; i++){
53. **if**(Keynumber < bound \* i && Keynumber >= bound \* (i - 1)) {
54. **return** i - 1;
55. }
56. }
57. **return** -1 ;
58. }
59. }
60. }

先将1、2、3上传到指定文件夹下：



执行代码后查看最终结果：



显然和样例答案相同，代码正确。

**（三）对给定的表格进行信息挖掘**

下面给出一个child-parent的表格，要求挖掘其中的父子辈关系，给出祖孙辈关系的表格。

输入文件内容如下：

|  |
| --- |
| child parent  Steven Lucy  Steven Jack  Jone Lucy  Jone Jack  Lucy Mary  Lucy Frank  Jack Alice  Jack Jesse  David Alice  David Jesse  Philip David  Philip Alma  Mark David  Mark Alma |

输出文件内容如下：

|  |
| --- |
| grandchild grandparent  Steven Alice  Steven Jesse  Jone Alice  Jone Jesse  Steven Mary  Steven Frank  Jone Mary  Jone Frank  Philip Alice  Philip Jesse  Mark Alice  Mark Jesse |

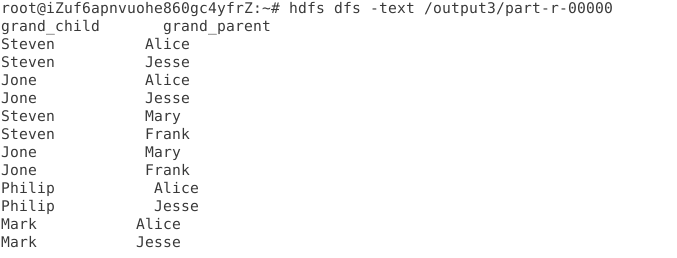
**代码：**

1. **import java.io.IOException;**
2. **import** java.util.ArrayList;
3. **import** java.util.List;
5. **import** org.apache.hadoop.conf.Configuration;
6. **import** org.apache.hadoop.fs.Path;
7. **import** org.apache.hadoop.io.\*;
8. **import** org.apache.hadoop.mapreduce.\*;
9. **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10. **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. **import** org.apache.hadoop.util.GenericOptionsParser;
13. **public** **class** Main {
14. **public** **static** **int** time = 0;
16. **public** **static** **class** Map **extends** Mapper<Object, Text, Text, Text> {
17. @Override
18. **public** **void** map(Object key, Text val, Context con) **throws** IOException, InterruptedException {
19. String line = val.toString();
20. String[] CAndP = line.split(" ");
21. List<String> list = **new** ArrayList<>(2);
22. **for** (String person : CAndP) {
23. **if** (!"".equals(person)) {
24. list.add(person);
25. }
26. }
27. **if** (!"child".equals(list.get(0))) {
28. String child = list.get(0);
29. String parent = list.get(1);
30. String relationType = "1";
31. con.write(**new** Text(parent), **new** Text(relationType + "+" + child + "+" + parent));
32. relationType = "2";
33. con.write(**new** Text(child), **new** Text(relationType + "+" + child + "+" + parent));
34. }
35. }
36. }
38. **public** **static** **class** Reduce **extends** Reducer<Text, Text, Text, Text> {
39. @Override
40. **public** **void** reduce(Text key, Iterable<Text> val, Context con) **throws** IOException, InterruptedException {
41. **if** (time == 0) {
42. con.write(**new** Text("grand\_child"), **new** Text("grand\_parent"));
43. time++;
44. }
45. List<String> grandChild = **new** ArrayList<>();
46. List<String> grandParent = **new** ArrayList<>();
47. **for** (Text ss : val) {
48. String s = ss.toString();
49. String[] relation = s.split("\\+");
50. String relationType = relation[0];
51. String child = relation[1];
52. String parent = relation[2];
53. **if** ("1".equals(relationType)) {
54. grandChild.add(child);
55. } **else** {
56. grandParent.add(parent);
57. }
58. }
59. **int** grandParentNum = grandParent.size();
60. **int** grandChildNum = grandChild.size();
61. **if** (grandParentNum != 0 && grandChildNum != 0) {
62. **for** (**int** m = 0; m < grandChildNum; m++) {
63. **for** (**int** n = 0; n < grandParentNum; n++) {
64. con.write(**new** Text(grandChild.get(m)), **new** Text(
65. grandParent.get(n)));
66. }
67. }
68. }
69. }
70. }
72. **public** **static** **void** main(String[] args) **throws** Exception {
73. Configuration conf = **new** Configuration();
74. conf.set("fs.defaultFS", "hdfs://localhost:9000");
75. Job job = Job.getInstance(conf, "判断爷孙");
76. job.setJarByClass(Main.**class**);
77. job.setMapperClass(Map.**class**);
78. job.setReducerClass(Reduce.**class**);
79. job.setOutputKeyClass(Text.**class**);
80. job.setOutputValueClass(Text.**class**);
81. FileInputFormat.addInputPath(job, **new** Path("/input/a.txt"));
82. FileOutputFormat.setOutputPath(job, **new** Path("/output3"));
83. System.exit(job.waitForCompletion(**true**) ? 0 : 1);
84. }
86. }

先将文件a上传到指定文件夹下：



执行代码后查看结果如下：



结果与预期相同，程序正确。

# 3. 实验总结

MapReduce编程就是在HDFS上用Map和Reduce编程，先Map把数据映射好，然后Reduce对数据进行操作，得到最终的结果。可以简单类比C++或者Java的map数据结构的Key-Value作用，但是Hadoop能处理更大的数据量。

第一题合并去重，很清楚。直接把数据当做key映射，value值可以直接空，那么重复加的自然就去重了。然后Reduce的时候把所有数据取出来，写到文件里，那么就完成了合并去重了。

第二题合并排序，然后还要给出每个数的排名。这个建议先百度，直接看代码有点看不懂，需要结合一点MapReduce的机制。因为key到reducer的分配是Partitioner决定的，虽然MapReduce的性质决定key是排序的，但是我分配到Partitioner不一样，那么我依次读Partitioner，每个Partitioner里的确是有序的，但是Partitioner之间的数不一定是有序的（例如[1,3,4],[2,5,6]），所以要想办法使得Partitioner之间的关系为升序。假设数值最大值为MAX\_INT，那么我们按照MAX\_INT给所有数分块，然后按照块的范围分给指定的Partitioner（），那么显然序号小的Partitioner里的key就比序号大的key小了。那么又由于MapReduce是按key排序的，并且此时Partitioner也是按key排序的，那么最终输出的key就是升序的。排名只要从1递增输出即可。

第三题给出父子，求爷孙关系。那么关键就是父值，如果一个值有父亲也有儿子，那么和它关联的两个值为爷孙关系。所以我们按照“父亲-儿子”、“儿子-父亲”再加上一个指定的标识符映射（我约定1表示value是key的儿子，2相反），这样我们读出来就知道key和value的关系了。然后Reduce的时候，假如当前的key值是Tom，那么我们把所有标识符为1的value存到child列表里，把标识符为2的value存到parent列表里，那么这两个列表里各取出一个值的任意二元组就构成一个爷孙关系，所以笛卡尔积组合一下输出就是答案。为了和题目答案格式契合，先输出“grand\_child grand\_parent”

总结就是MapReduce很有趣，映射很有意思。第二题需要想的比较多，第三题需要写的比较多。