

Harry Potter的魔法世界

小组成员

201830059 陈思睿 1310459235@qq.com

201220098 杨林 1479016371@qq.com

201830122 邹忠奇 617397223@qq.com

程序设计

任务一

输入

- 文件夹，内有未分词的小说
- 人物名字列表，包括全名和简称

输出

- 提取出每段话出现的人名

实现

由于我在任务1中就进行了人名统一，我的任务1的输出格式形如

PERSON_A,COUNT_A;PERSON_B,COUNT_B;...(e.g Harry,2;Ron 3;Hermione 1;)

Mapper

public static class GetNamesMapper extends Mapper<Object, Text, Text, Text> 为了便于
统一人名,我实现了 PersonWritableComparable 类,该类在任务3中也有用到.

```
public class PersonWritableComparable implements WritableComparable {  
  
    private int nameWordNum;  
    private String[] fullName;  
    //some methods to implement WritableComparable  
}
```

Mapper.setup()

`setup` 的任务是建立人物全名集合,以及人物简称到一系列全名的映射表。

我使用 `context.getLocalCacheFile()` 来读取人名列表。按单词的数量从大到小的顺序遍历人名列表, 若一个人名的单词数为1且存在一个已经遍历到的人名包含它, 则将该人名归类为简称, 否则归类为全名。此处时间复杂度 $\Theta(\text{人名表长} \cdot \log(\text{人名表长}))$

一个简称可能对应不止一个全名 (例如姓氏, 人名列表中的Potter被视为简称, 它对应的全名有Harry Potter, James Potter和Lily Potter), 若一个全名包含该简称, 则该全名在该简称对应的全名集合中。简称到全名集合的映射表可以通过对每个简称, 遍历所有全名的方式进行, 总时间复杂度为 $\Theta(\text{人名表长}^2)$ 。虽然也有 $\Theta(n \log n)$ (n 为人名表长) 的做法, 但由于人名列表的长度只有240, 此处不是时间瓶颈。

Mapper.map()

我对匹配到的人物简称进行了特殊处理, 以加强统计的准确性。

`map(Object key, Text value, Context context)`, 其中 `value` 的值为原小说的某一段落。

将`value`转换成字符串, 暴力匹配人名。

- 若匹配到了全名, 则直接将此人的计数器+1
- 若匹配到了简称
 - 若简称只对应一个全名, 则将对应的人的计数器+1
 - 否则将该简称放入waitlist中

然后遍历waitlist, 若该简称对应的全部人名中有且仅有一个出现在了此段落中, 则视作该简称指代这个唯一出现过的人, 将此人的计数器+1 (例如: Ron Weasley says to Harry Potter: "Hello, Potter!", 此处的Potter应指代Harry Potter而非James Potter或Lily Potter)。否则 (都未出现或出现不止一个人) 无法判断该简称指谁, 计数器都不变。

最后将该段落出现的所有人以及出现的次数写入 `context`, 形如
`PERSON_A,COUNT_A;PERSON_B,COUNT_B;...`(e.g Harry,2;Ron 3;Hermione 1;)

时间复杂度 $\Theta(n \log p)$, 其中n为段落长度, p为人物表长度。

Reducer

```
public static class GetNamesReducer extends Reducer<Text, Text, Text>
```

Mapper干了绝大部分的活, Reducer无需特殊处理。

运行结果:

application_1678703754602_7341	2023fg31	GetNames	MAPREDUCE	MapReduce	0	ZUZ3	Sat Jul 15 19:32:51 19:32:51 +0800	Sat Jul 15 19:33:33 +0800 2023	FINISHED	SUCCEEDED	N/A
--------------------------------	----------	----------	-----------	-----------	---	------	---	---	----------	-----------	-----

文件 - /user/2023fg31/lab5/t1out/part-r...

Page 1 of 683 ◀ ▶ ▷ C

```
Harry Potter,1;Petunia Dursley,1;Dudley Dursley,1;
Petunia Dursley,1;
Dudley Dursley,1;
Dudley Dursley,1;
Harry Potter,1;
Harry Potter,1;
Harry Potter,1;
Dudley Dursley,1;
Dudley Dursley,1;
Harry Potter,1;Dudley Dursley,1;
Dudley Dursley,1;
Harry Potter,1;
Harry Potter,1;
Voldemort,1;
Harry Potter,1;
```

取消

下载

任务二

通过任务一的处理，我们获得了如以下格式的输出

```
Name1, d1; Name2, d2; Name3, d3; ...; Namei, di;
```

其中 `Namei` 表示人名， `di` 表示该人名在某一行里出现的次数。

对输入的每一行，遍历其中的人名与出现次数。生成人名对及对应的出现次数乘积 $\langle (Namei, Namej), di * dj \rangle$ ，其中 $i \neq j$ 。

再把相同的人名对的出现次数加起来，得到最终的人名对及其对应的出现次数。

Mapper

输入 key: 行号

输入 value: 以分号隔开的人名及出现次数, 人名和出现次数之间用逗号分开

```

void map(key,value, Mapper<LongWritable, Text, Text, LongWritable>.Context context){
    // 把人名及出现次数分隔开
    nameAppearList = value.split(";\\"s*");

    // 两轮遍历产生所有可能的人名对
    for (i = 0; i < nameAppearList.length; i++) {
        // 人名1 及出现频率
        name1 = nameAppearList[i].split(",")[0];
        t1 = nameAppearList[i].split(",")[1];
        for (int j = 0; j < nameAppearList.length; j++) {
            if (j == i)
                continue;

            // 人名2 及出现频率
            name2 = nameAppearList[j].split(",")[0];
            t2 = nameAppearList[j].split(",")[1];

            // (name1, name2) 和 (name2, name1)对偶, 都需要输出.
            context.write(name1 + name2, int(t1) * int(t2));
            context.write(name2 + name1, int(t1) * int(t2));
        }
    }
}

```

输出 key: 人名对

输出 value: 这个人名对在某一段的出现次数

Reducer

输入 key: 人名对

输入 values: 人名对在所有段的出现次数

```

void reduce(key, values, Reducer<Text, LongWritable, Text,
LongWritable>.Context context) {
    sum = 0L;
    // 遍历values, 把所有的出现次数累加
    for(v : values){
        sum += v.get();
    }
    context.write(key, sum);
}

```

输出 key: 人名对

输出 value: 人名对的总出现次数

运行结果

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1678703754602_7342	2023fg31	Feature Extraction	MAPREDUCE	MapReduce	0	Sat Jul 15 19:35:26 +0800 2023	Sat Jul 15 19:35:26 +0800 2023	Sat Jul 15 19:36:11 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	History	0	

文件 - /user/2023fg31/lab5/t2out/part-r-00000

Page 1 of 109 ◀ ▶ ▷ C

```
<Albus Dumbledore,Alicia Spinnet> 4
<Albus Dumbledore,Amos Diggory> 2
<Albus Dumbledore,Angelina Johnson> 2
<Albus Dumbledore,Bane> 4
<Albus Dumbledore,Bertha Jorkins> 14
<Albus Dumbledore,Bill Weasley> 18
<Albus Dumbledore,Bloody Baron> 2
<Albus Dumbledore,Buckbeak> 18
<Albus Dumbledore,Cedric Diggory> 24
<Albus Dumbledore,Cho Chang> 6
<Albus Dumbledore,Colin Creevey> 6
<Albus Dumbledore,Cornelius Fudge> 284
<Albus Dumbledore,Crookshanks> 6
<Albus Dumbledore,Dean Thomas> 12
<Albus Dumbledore,Dobby> 62
```

取消

下载

输出了每一对人物之间的同现关系

任务三

任务三的内容是根据任务二得到的输入对 `<PersonA, PersonB>Count` ,对于每个人,计算其他所有人占其总社交的比例.

任务比较简单。我新建了 `private class NeighborWritable` , 域为to (人物) 和count (出现次数) 。

```
private static class NeighborWritable implements Writable {
    PersonWritableComparable to;
    long count;
    //some methods to implement Writable
}
```

Mapper

```
public static class GraphFileGenMapper extends  
Mapper<Object, Text, PersonWritableComparable, NeighborWritable>
```

在mapper中，对于输入 `<Person_A,Person_B>Count`，执行 `context.write(Person_A,new NeighborWritable(Person_B,Count))`。

Reducer

```
public static class GraphFileGenReducer extends
```

`Reducer<PersonWritableComparable, NeighborWritable, Text, Text>` 在reducer中，统计每个人与所有其他共现的总次数，将每个人的共现次数除以总次数，得到每个人的比例。每个人出现的次数可以用 `HashMap` 统计.

运行结果：

application_1678703754602_7343	2023fg31	GraphFileGen	MAPREDUCE	MapReduce	0	2023	Sat Jul 15	Sat Jul 15	Sat Jul 15	FINISHED	SUCCEEDED	N/A
							19:37:41	19:37:41	19:38:24			
							+0800	+0800	+0800			
							2023	2023	2023			

文件 - /user/2023fg31/lab5/t3out/part-r-...

Page 1 of 112 ◀ ▶ ⌂

```
Albus Dumbledore [Molly Weasley,0.0022876751501287|Nearly Headless  
Nick,0.0017157563625965|Fawkes,0.0057191878753217|Professor Sprout,0.0022876751501287|Ginny  
Weasley,0.0040034315127252|Lee Jordan,0.0002859593937661|Professor Minerva  
Mcgonagall,0.0042893909064913|Professor Severus  
Snape,0.0686302545038605|Buckbeak,0.0025736345438948|Filch,0.0042893909064913|Amos  
Diggory,0.0002859593937661|James Potter,0.0054332284815556|Hermione Granger,0.0448956248212754|Harry  
Potter,0.3963397197597941|George Weasley,0.0057191878753217|Rubeus Hagrid,0.0408921933085502|Salazar  
Slytherin,0.0062911066628539|Igor Karkaroff,0.0091507006005147|Hedwig,0.0025736345438948|Seamus  
Finnigan,0.0020017157563626|Helga Hufflepuff,0.0028595939376609|Tom  
Riddle,0.0165856448384329|Scabbers,0.0002859593937661|Professor Grubbly-Plank,0.0008578781812983|Fat  
Friar,0.0020017157563626|Bane,0.0005719187875322|Draco Malfoy,0.0214469545324564|Dudley  
Dursley,0.0020017157563626|Katie Bell,0.0008578781812983|Bloody Baron,0.0002859593937661|Mr Crouch,0.012
```

取消

下载

任务四

任务四分为3个Mapreduce模块，Init,Work和Main

Init

Init为对task3输出的图的简单格式处理。

task3的输出格式为 Person [NeighborA, proportionA|NeighborB, proportionB|...]

Init后的格式为 Person ;pageRank值(初始为 $\frac{1}{PersonCount}$)；总人数;NeighborA, proportionA, NeighborB, proportionB, ...

Work

work为一次pagerank迭代。

Map

```
public static class PRWorkMapper extends Mapper<Object, Text,  
PersonWritableComparable, Text>
```

PageRank算法的公式为 $PRVal[u] = \frac{1-D}{N} + D \cdot \sum_{v \in pred(u)} PRVal[v] \cdot proportion(v, u)$, 转换成MapReduce的写法就是 for w : successors of u, $PRVal[w] += D * PRVal[u] * proportion(u, w)$

对于每条value，形如 Person ;PRVal;N;NeighborA,proportionA;NeighborB,proportionB,...

1. 遍历所有的 Neighbor，对每个Neighbor执行 context.write(Now_Neighbor, "+" + D * PRVal * Now_proportion)。
2. 将这行value原封不动地写入 context，key值为这条value对应的Person。 (因为我们需要在输出中保留这张图)

Combine

```
public static class PRWorkCombiner extends  
Reducer<PersonWritableComparable, Text, PersonWritableComparable, Text>
```

- 若value以"+"开头，则说明是PRVal的计算信息，统计和。
- 否则是此人的信息列表，直接写入context
- 最后将所有PRVal计算信息的和写入context

Reduce

```
public static class PRWorkReducer extends  
Reducer<PersonWritableComparable, Text, Text, Text>
```

将key对应的人的PRVal初始值设为0

遍历所有value：

- 若遇到以 "+" 开头的 value, 说明这个 value 是关于此人 PRVal 的计算信息 (即 Map 中的 1.) , 将此人的 PRVal 加上该值。
- 否则, 说明这个 value 是此人的信息列表 (即 Map 中的 2.) , 暂时记录, 并且读取 N 的值。

将 PRVal 加上 $\frac{1-D}{N}$, 修改此人的信息列表, 输出到 context

Main

含有排序 MapReduce。

主要功能是控制流: 先执行 init, 然后按照传进来的参数迭代 work, 最后执行排序 MapReduce.

在平台上迭代 3 次的运行结果:

文件 - /user/2023fg31/lab5/t4out/result/...

```
Harry Potter 0.219300593639741
Ron Weasley 0.08814962117755776
Hermione Granger 0.07199040507244442
Albus Dumbledore 0.03874400546308296
Professor Severus Snape 0.029188050669228766
Rubeus Hagrid 0.027997698752684753
Sirius Black 0.02571823463227528
Draco Malfoy 0.025143210049819516
Fred Weasley 0.0186573746687775
Godric Gryffindor 0.01724442925094747
George Weasley 0.017029453450437174
Ginny Weasley 0.015281883337437638
Neville Longbottom 0.014157817186632842
Voldemort 0.013313135823675663
Professor Lupin 0.011055668689240345
```

这是平台跑 5 轮的结

果, 可以看出与迭代 3 轮的结果相差不多:

Harry Potter	0.21600444437353078
Ron Weasley	0.08978671441400816
Hermione Granger	0.07297366562883757
Albus Dumbledore	0.03907511587408613
Professor Severus Snape	0.029236115191738177
Rubeus Hagrid	0.028202640006000188
Sirius Black	0.025753412642658142
Draco Malfoy	0.02554660583759411
Fred Weasley	0.018754744494114028
Godric Gryffindor	0.01722720988436231
George Weasley	0.017059417394916822
Ginny Weasley	0.015472035610874534
Neville Longbottom	0.014327876742687677
Voldemort	0.013313935550881603
Professor Lupin	0.0110625304855317

问题：迭代5轮时有时平台报错，同样的程序每次跑报的错还不一样)

有时是超时（这个任务的时间复杂度不高，只有遍历的时间复杂度，Combiner也加了，但还是超时）：

```
2023-07-16 01:57:55,296 INFO mapreduce.Job: Task Id : attempt_1678703754602_7448_r_000000_2, Status : FAILED
Container launch failed for container_1678703754602_7448_01_000004 : org.apache.hadoop.yarn.exceptions.YarnException: Unauthorized request to start container.
This token is expired. current time is 1689488187142 found 168944475011
Note: System times on machines may be out of sync. Check system time and time zones.
```

有时是No Route to Host:

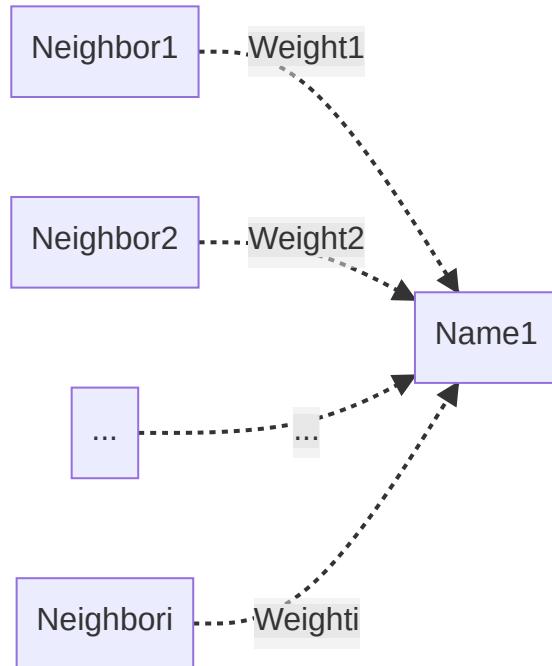
```
2023-07-16 11:37:19,469 INFO mapreduce.Job: Job job_1678703754602_7570 failed with state FAILED due to: Application application_1678703754602_7570 failed 2 times due to Error launching appattempt_1678703754602_7570_000002. Got exception: java.net.NoRouteToHostException: No Route to Host from master001/192.168.1.1 to slave004:33540 failed on socket timeout exception: java.net.NoRouteToHostException: No route to host; For more details see: http://wiki.apache.org/hadoop/NoRouteToHost
at sun.reflect.GeneratedConstructorAccessor165.newInstance(Unknown Source)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
```

任务五

通过任务三的处理，我们得到了一个人物关系图，这个人物关系图的每一行形式如下

Name1 [Neighbor1,Weight1 Neighbor2,Weight2 ... Neightbori,Weighti]
--

在这一行中，表示了如图信息：



也就是说这一行记录了一个结点及其入边的信息.

整个算法过程分成三步:

第一步: 给每个结点打上一个标签, 将入边图转换成出边图. 进入第二步.

第二步: 在出边图, 对每个结点, 对其所有的出边邻居派发结点的人名, 标签和权重. 每个结点对收到的所有人名, 标签和权重信息, 选择总权重最大的标签作为自己的新标签, 并生成一张入边图. 进入第三步.

第三步: 将入边图重新转化成出边图, 重复第二步.

第一步--初始化

Mapper

输入 key: 结点名

输入 value: 结点的入边及对应权重

```

void map(key,value, Mapper<Text, Text, Text, Text>.Context context){
    //根据 '['，'|'，']' 分隔符将入边分隔开.
    inEdges = value.toString().split("[\\[\\]\\]|]");
    for (innodes : inEdges) {
        String node_name = innodes.split(",")[0];
        String weight = innodes.split(",")[1];
        // 将边转置, 由 Neighbor-->Key 转变为 Key-->Neighbor
        context.write(node_name, key + "," + weight);
    }
}

```

输出 key: 结点名

输出 value: key的出边结点及权重

Reducer

输入 key: 结点名

输入 values: 结点的每一个出边结点及对应权重.

```

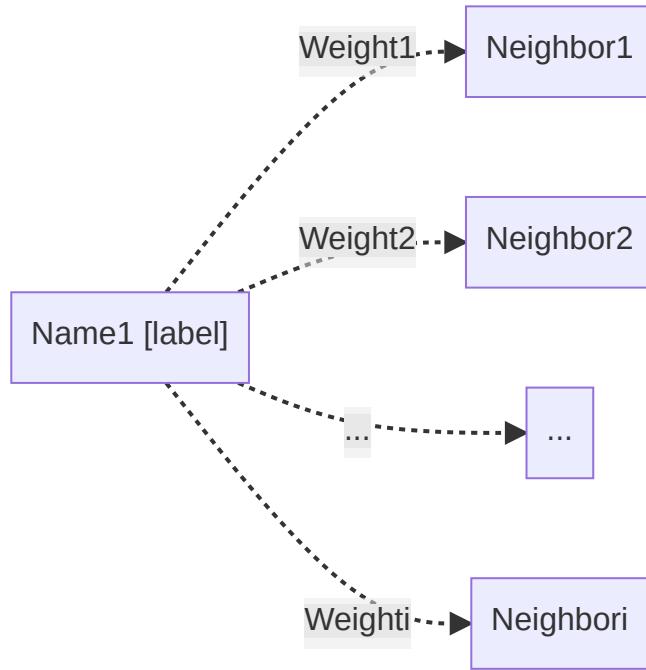
void reduce(key, values, Reducer<Text, Text, Text, Text>.Context context){
    valueBuilder = new StringBuilder();
    for (v : values) {
        valueBuilder.append(v.toString());
        valueBuilder.append(";");
    }
    context.write( key + "[" + key + "]", valueBuilder.toString());
}

```

输出 key: 结点名及其标签

输出 values: 结点的出边结点和对应的权重

形象地说,图的表达被转化成了这种形式:



第二步--传播

这一步为每个结点选择自己的新标签

Mapper

输入 key: 人名及其标签

输入 value: 出边邻居和边上的权重

```

void map(key, value, Mapper<Text, Text, Text, LPAPPropagateValue>.Context
context) {
    // 把输入的Key切割成人名和其标签
    key_name = key.split("[\\[\\]]") [0];
    key_label = key.split("[\\[\\]]") [1];
    // 根据分号把出边邻居分开
    outNodes = value.split(";");
    for (outNode : outNodes) {
        // 根据逗号把出边邻居人名和权重分开
        nodeName = outNode.split(",") [0];
        nodeWeight = outNode.split(",") [1];
        context.write(nodeName, {key_name, key_label, nodeWeight});
    }
}

```

输出 key: 出边邻居人名

输出 value: 结点的人名, 标签和边上的权重

Reducer

输入 key: 结点人名

输入 values: 结点所有入边邻居人名, 标签和边上的权重

```
void reduce(key, values, Reducer<Text, LPAPropagateValue, Text, Text>.Context context) {
    List<String> inNames = new ArrayList<>();
    List<Double> inWeights = new ArrayList<>();
    Map<String, Double> label_map = new HashMap<>();
    for (v : values) {
        // 把入边邻居和权重收集起来
        inNames.add(v.name);
        inWeights.add(v.weight);
        // 记录标签及其对应的权重
        if (label_map.containsKey(v.label)) {
            label_map.put(label, label_map.get(label) + weight);
        } else {
            label_map.put(label, weight);
        }
    }
    // 找到权重最大的label
    max_label = Collections.max(label_map.entrySet(),
        Map.Entry.comparingByValue()).getKey();
    // 将入边人名和权重重新组合
    valueBuilder = new StringBuilder();
    for (int i = 0; i < inNames.size(); i++) {
        valueBuilder.append(inNames.get(i) + "," + inWeights.get(i) + ";");
    }
    // 输出
    context.write(key + "[" + max_label + "]", valueBuilder.toString());
}
```

输出 key: 结点人名及其新的标签

输出 value: 结点的入边邻居人名及边上的权重

在这里, 出边图又变成了入边图, 所以下一步需要将入边图重新转化成出边图

第三步--图重建

这一步的主要目的就是将第二步的输出重新组合成第二步的输入的格式.

Mapper

输入 key: 结点人名及其标签

输入 value: 结点的入边邻居人名及边上的权重

```
void map(key, value, Mapper<Text, Text, Text, Text>.Context context){  
    // 将结点人名和标签分开  
    key_name = key.split("\\\\[\\\\])[0];;  
    key_label = key.split("\\\\[\\\\])[1];  
    // 遍历入边邻居  
    String[] innodes = value.split(";;");  
    for (String innode : innodes) {  
        // 将入边邻居及边上的权重根据逗号分开  
        node_name = innode.split(",")[0];  
        weight = innode.split(",")[1];  
        // 边的形式由 value --> key 转变为 key --> value  
        context.write(node_name, key_name + "," + weight);  
    }  
    // 最后把结点的标签也输出，因为仅仅之用了结点人名作为key，  
    // 需要把结点人名对应的新标签也输出让生成的出边图中的结点知道自己的新标签是什么  
    context.write(key_name, key_label);  
}
```

输出 key: 结点人名

输出 value: 结点的出边邻居及对应的权重, 或者结点的新标签

Reducer

输入 key: 结点人名

输出 values: 结点的出边邻居及对应的权重, 或者结点的新标签

```
void reduce(key, values, Reducer<Text, Text, Text, Text>.Context context) {  
    // 用来存储key 和value  
    valueBuilder = new StringBuilder();  
    keyBuilder = new StringBuilder();  
    keyBuilder.append(key.toString());  
    for (v : values) {  
        // 如果这是一个出边邻居和权重， 那么加入value  
        if (v_str.contains(",")) {  
            valueBuilder.append(v_str + ";");  
        }  
        //否则这是结点的标签  
        else {  
  
            keyBuilder.append("[ " + v_str + " ]");  
        }  
    }  
    context.write(keyBuilder.toString(), valueBuilder.toString());  
}
```

输出 key: 结点人名和标签

输出 values: 结点的所有出边邻居及对应的权重

运行结果

application_1678703754602_7377	2023fg31	LPA Rebuild	MAPREDUCE	MapReduce	0	Sat Jul 15 20:35:15 +0800 2023	Sat Jul 15 20:35:15 +0800 2023	Sat Jul 15 20:39:01 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7376	2023fg31	LPA Propagate	MAPREDUCE	MapReduce	0	Sat Jul 15 20:34:11 +0800 2023	Sat Jul 15 20:34:11 +0800 2023	Sat Jul 15 20:35:08 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7375	2023fg31	LPA Rebuild	MAPREDUCE	MapReduce	0	Sat Jul 15 20:33:24 +0800 2023	Sat Jul 15 20:33:24 +0800 2023	Sat Jul 15 20:34:08 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7372	2023fg31	LPA Propagate	MAPREDUCE	MapReduce	0	Sat Jul 15 20:32:18 +0800 2023	Sat Jul 15 20:32:18 +0800 2023	Sat Jul 15 20:33:20 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7371	2023fg31	LPA Init	MAPREDUCE	MapReduce	0	Sat Jul 15 20:31:27 +0800 2023	Sat Jul 15 20:31:27 +0800 2023	Sat Jul 15 20:32:12 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7370	2023fg31	PageRankinit	MAPREDUCE	MapReduce	0	Sat Jul 15 20:28:31 +0800 2023	Sat Jul 15 20:28:53 +0800 2023	Sat Jul 15 20:32:45 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7365	2023fg31	LPA Rebuild	MAPREDUCE	MapReduce	0	Sat Jul 15 20:12:08 +0800 2023	Sat Jul 15 20:12:08 +0800 2023	Sat Jul 15 20:14:38 +0800 2023	KILLED	KILLED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7364	2023fg31	LPA Propagate	MAPREDUCE	MapReduce	0	Sat Jul 15 20:08:07 +0800 2023	Sat Jul 15 20:11:27 +0800 2023	Sat Jul 15 20:12:03 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7359	2023fg31	LPA Rebuild	MAPREDUCE	MapReduce	0	Sat Jul 15 20:01:06 +0800 2023	Sat Jul 15 20:04:26 +0800 2023	Sat Jul 15 20:08:01 +0800 2023	FINISHED	FAILED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7358	2023fg31	LPA Propagate	MAPREDUCE	MapReduce	0	Sat Jul 15 20:00:09 +0800 2023	Sat Jul 15 20:00:10 +0800 2023	Sat Jul 15 20:01:02 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1678703754602_7357	2023fg31	LPA Init	MAPREDUCE	MapReduce	0	Sat Jul 15 19:56:22 +0800	Sat Jul 15 19:56:22 +0800 2023	Sat Jul 15 20:00:07 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

文件 - /user/2023fg31/lab5/t5out/temp/r1/propag...

Page 1 of 111

Albus Dumbledore[Harry Potter] Bertha Jorkins,0.0020017157563626;Bane,5.719187875322E-4;Angelina Johnson,2.859593937661E-4;Winky,0.0037174721189591;Mr Ollivander,5.719187875322E-4;Mr Crouch,0.0120102945381756;Parvati Patil,5.719187875322E-4;Molly Weasley,0.0022876751501287;Peeves,0.0020017157563626;Madam Pomfrey,0.0051472690877895;Percy Weasley,0.0074349442379182;Madam Maxime,0.0054332284815556;Mad-eye Moody,0.0160137260509008;Peter Pettigrew,5.719187875322E-4;Ludo Bagman,0.0045753503002574;Neville Longbottom,0.006863025450386;Petunia Dursley,0.0014297969688304;Lily Potter,0.0020017157563626;Pigwidgeon,5.719187875322E-4;Lee Jordan,2.859593937661E-4;Porfessor Trelawney,0.0040034315127252;Lavender Brown,2.859593937661E-4;Katie Bell,8.578781812983E-4;Professor Flitwick,0.0014297969688304;Nicolas Flamel,0.0022876751501287;James Potter,0.0054332284815556;Professor Grubbly-Plank,8.578781812983E-4;Igor Karkaroff,0.0091507006005147;Professor Lupin,0.0077209036316843;Hermione Granger,0.0448956248212754;Professor Miner

[取消](#)

[下载](#)

程序运行方式

任务一

运行指令:

```
yarn jar xxx.jar org.GetNames.GetNames path/person_name_list.txt input_path
output_path
```

第一个运行参数表示人名列表, 第二个参数表示输入目录,第三个参数表示输出目录

输入格式: 在输入目录下为哈利波特全集, 注意, person_name_list.txt文件不能和哈利波特全集在同一个目录下.

输出格式: 每一行代表一段话里的人名人名出现次数, 形如 Person_a, Count_a;Person_b, Count_b;...

任务二

运行指令:

```
yarn jar xxx.jar org.FeatureExtraction.FeatureExtraction input_path output_path
```

输入格式: 任务一的输出格式

输出格式: 每一行代表一个人名对,后面跟着同现次数, 形如 <Person_a, Person_b> Count , 表示这两个人在文中的同现次数

任务三

运行指令:

```
yarn jar xxx.jar org.GraphFileGen.GraphFileGen input_path output_path
```

输入格式: 任务二的输出格式

输出格式: 每一行有一个人名, 后面跟着与其有同现关系的人名列表和对应的权重, 形如 Person [Neighbor1, Weight1|Neighbor2, Weight2|...]

任务四

运行指令

```
yarn jar xxx.jar org.PageRank.PageRankMain input_path output_path iterate_times
```

第三个参数表示任务运行的迭代次数

输入格式: 与任务三的输出格式相同

输出格式: 在output_path目录下, 最终会有一个result文件夹, 文件夹里的输出结果即为最终的结果, 里面是根据数值从大到小排列好的人名.

任务五

运行指令

```
yarn jar xxx.jar org.LPA.LPAMain input_path output_path iterate_times
```

输入格式: 与任务三的输出格式相同

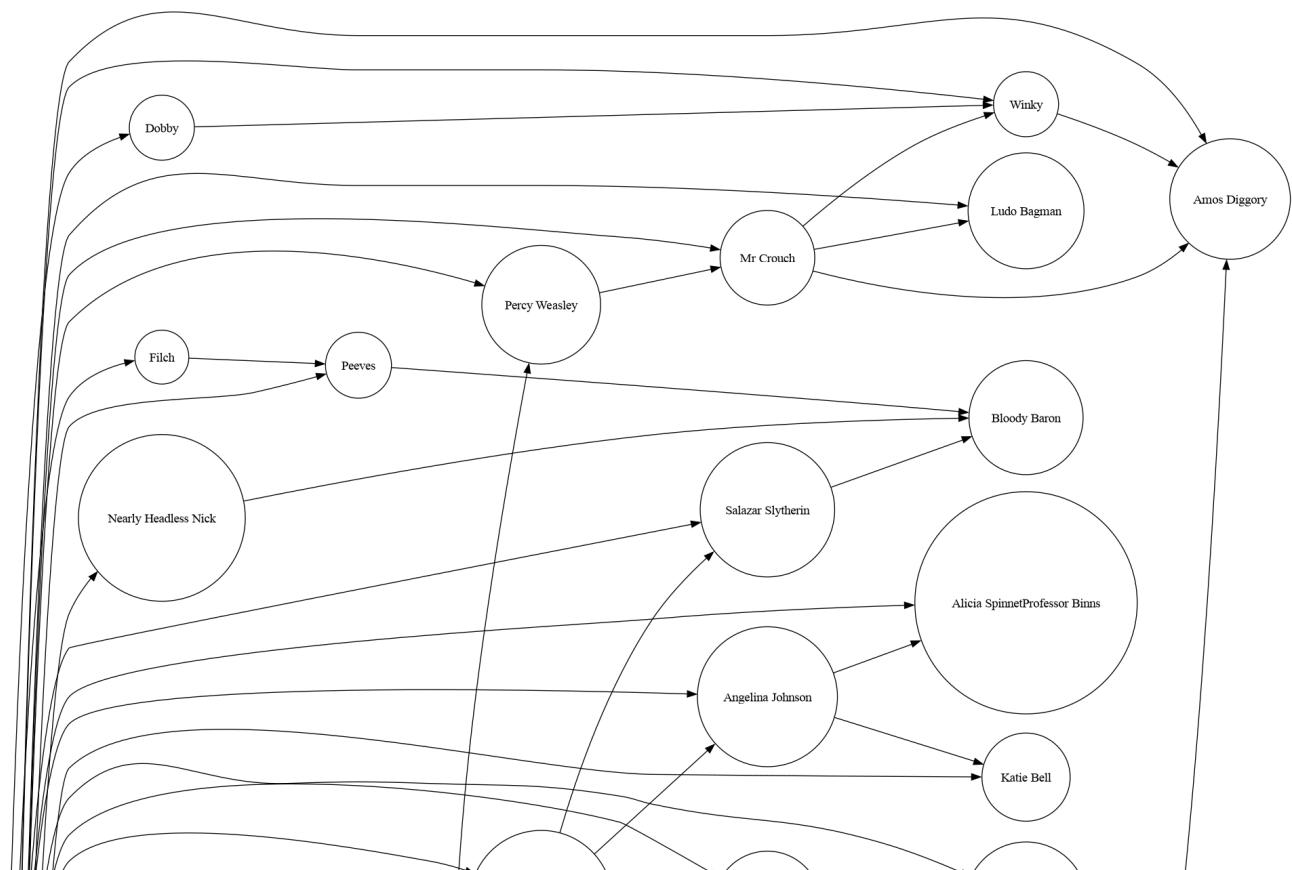
输出格式: 在output_path目录下, 会有各次迭代的输出, 用 r0, r1, r2 表示出来了, 在 propagate 目录下的是入边图, rebuild 目录下的是出边图.

程序运行结果分析

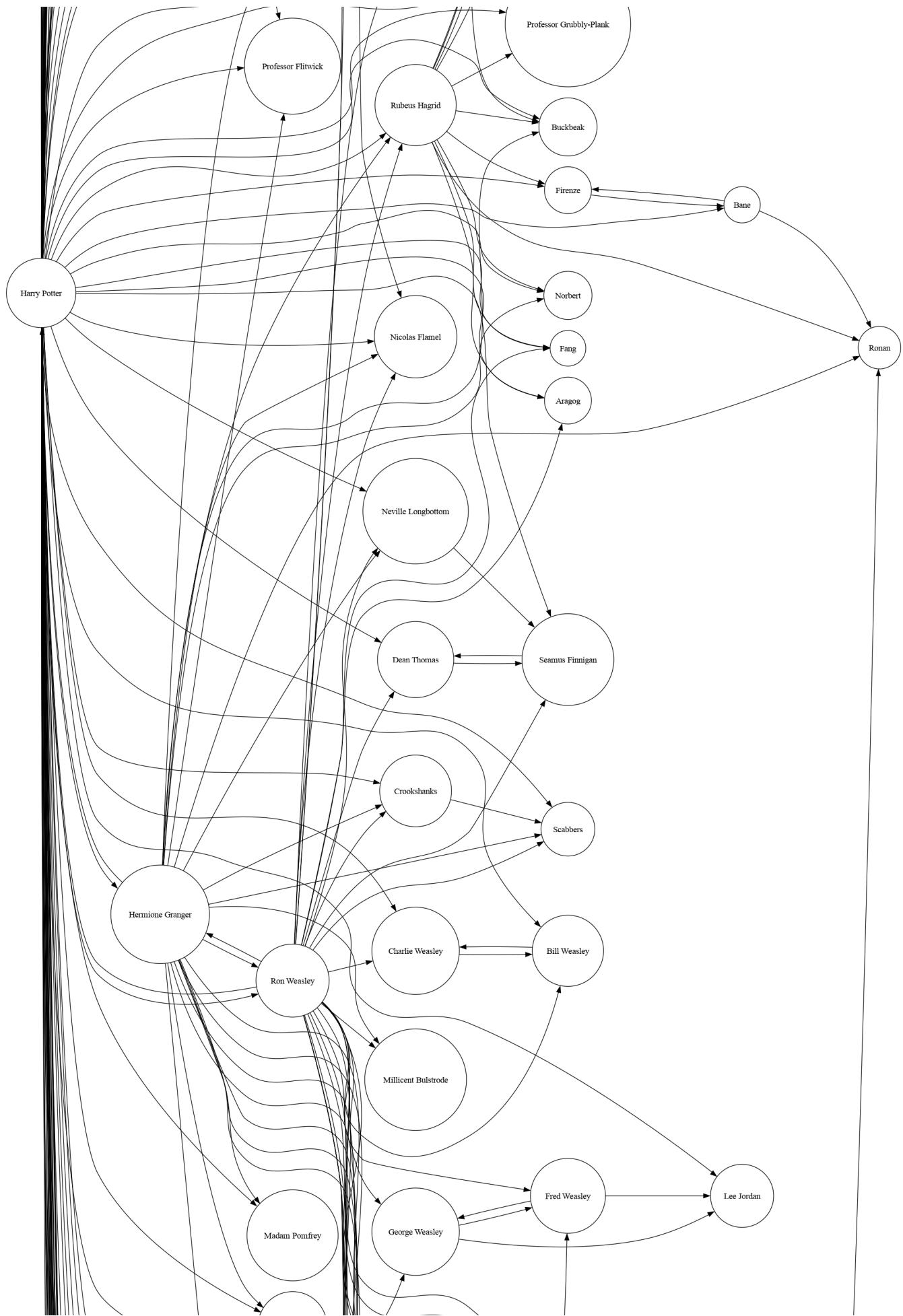
任务一, 任务二的输出结果比较抽象, 不适合人眼分析, 这里从任务三开始.

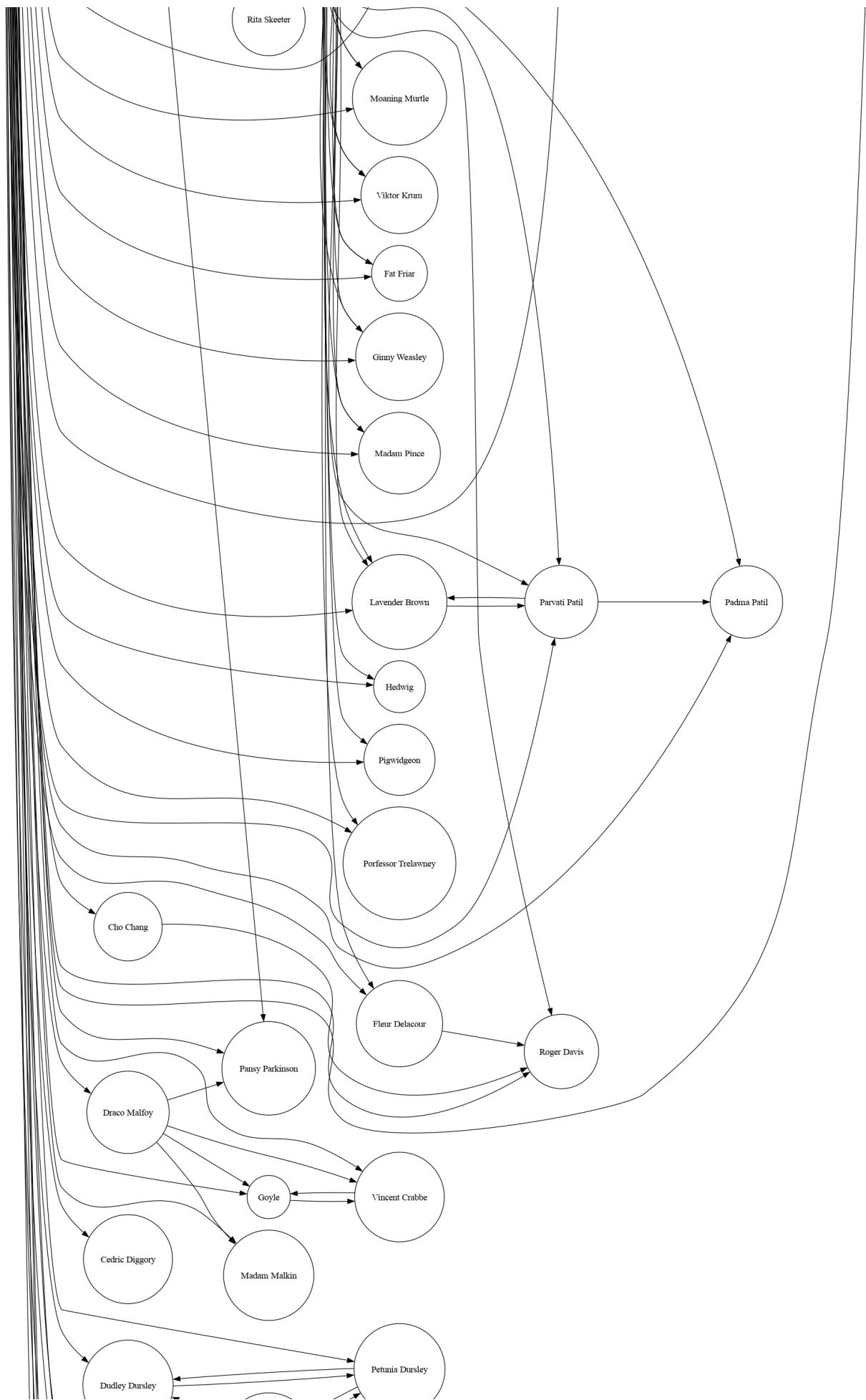
任务三

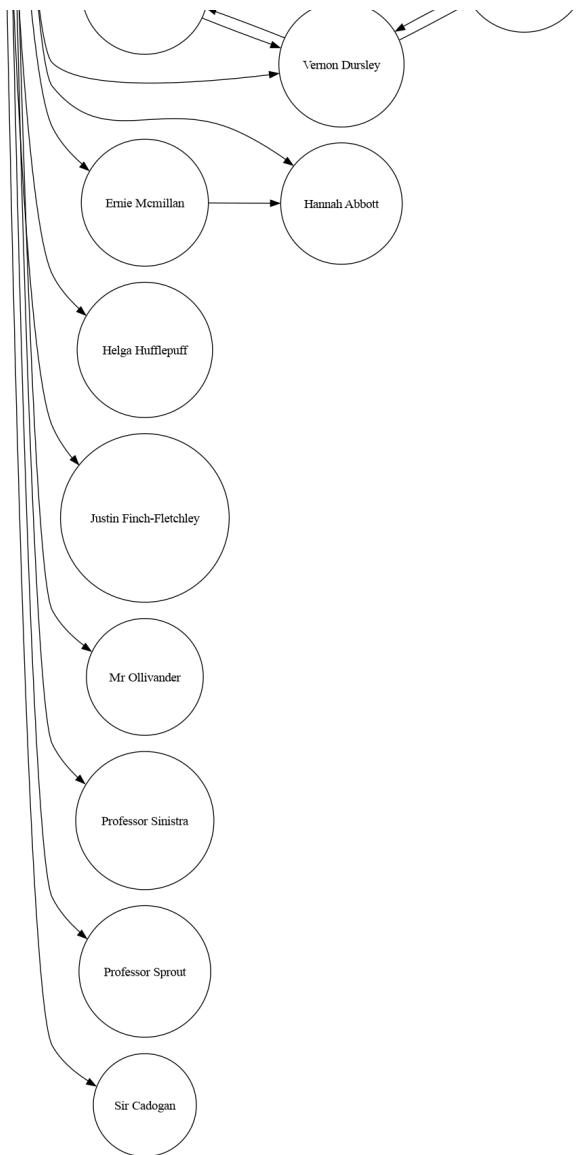
这个任务的负责将人物关系图画出来, 我根据任务三的输出, 画了一张图, 对于一条边, 如果边上的权重大于0.1那么这条边将会在图上画出. 不把所有边都画出来的原因是画不出来.











这张图很大, 因为包括了任务一筛选出来的一百多个结点和他们之间的边. 在图中我们可以注意到, 基本上每一个结点都有一条来自"Harry Potter"的入边, 这说明Harry Potter对每一个人物的关系比重都大于0.1, 这是符合我们对这部小说的印象的, 因为这部小说就叫"Harry Potter".

任务四

文件 - /user/2023fg31/lab5/t4out/result/...

Harry Potter	0.219300593639741
Ron Weasley	0.08814962117755776
Hermione Granger	0.07199040507244442
Albus Dumbledore	0.03874400546308296
Professor Severus Snape	0.029188050669228766
Rubeus Hagrid	0.027997698752684753
Sirius Black	0.02571823463227528
Draco Malfoy	0.025143210049819516
Fred Weasley	0.0186573746687775
Godric Gryffindor	0.01724442925094747
George Weasley	0.017029453450437174
Ginny Weasley	0.015281883337437638
Neville Longbottom	0.014157817186632842
Voldemort	0.013313135823675663
Professor Lupin	0.011055668689240345

在这个结果能看到, PageRank值最高的自然是哈利波特, 接下来的两位主角三人组成员的PageRank值也远高于剩下的人.之后的人都是哈利波特的老师,同学和亲人.排名最靠前的反派,也是小说中最大的反派,仅排在第14名,说明小说的内容以校园生活为主,打反派所占比例不大.

任务五

在实验三的图中我们也看到了, 几乎每一个人身上Harry Potter的权重都不小, 导致在任务五当中, 标签传播多跑几轮, 就全部只剩Harry Potter了, 在平台上我只跑了两轮, 最终就只剩下了两个标签--"Harry Potter" 和 "Ron Weasley", 某种程度上也反应了本书的写作手法, 本书主要以Harry Potter的视角为主, 展示Harry Potter的魔法世界.

总结

功能

通过对任务三的分析我们能够了解到小说中人物与人物之间的关系比重, 全面的刻画了人物关系谱图.

在任务三的基础上做PageRank算法我们能够进一步的以程序的视角发现小说中的重要人物.

在任务三的基础上做标签传播算法我们期望能够发现人物之间的社区关系, 或者说子图关系, 但是由于本文的写作手法的问题, 我们最终得出的只有一个标签那就是Harry Potter.

改进

最令人不满意的部分应该是任务五中标签传播的结果太过于收敛, 或许在已知Harry Potter作为主角之后, 我们可以尝试着将Harry Potter 从人物关系图上去掉, 在没有Harry Potter的图上运行标签传播算法, 或许能够进一步的给小说人物进行分类.