

# 金庸的江湖——金庸武侠小说中的人物关系挖掘

组长：韩畅，组员：李展烁、王一之、闫旭芑

2020 年 7 月 29 日

## 1 实验规划与设计

### 1.1 任务分配

171860551, 韩畅：组长, 算法设计与实验规划, 任务六

171860550, 王一之：算法设计与实验规划, 任务四

171860549, 闫旭芑：算法设计与实验规划, 任务五

171840565, 李展烁：算法设计与实验规划, 任务三

### 1.2 任务要求

### 1.3 设计思路

## 2 实验实现

### 2.1 任务一

todo

### 2.2 任务二

todo

### 2.3 任务三

todo

### 2.4 任务四: 基于人物关系图的 PageRank 计算

#### 2.4.1 PageRank 算法介绍

PageRank, 又称网页排名, 名字源于 google 创始人之一的 Larry Page, 是 Google 公司所使用的对与网页重要性排序的算法。

PageRank 通过网页之间的超链接评价网页重要性, 它的基本思想是:

- 1) 如果一个网页被多个网页所指向, 则该网页比较重要
- 2) 如果一个重要的网页指向另一个网页, 则另一个网页也比较重要

该算法模拟一个上网者, 随机打开一个网页, 之后随机点击该网页的链接, 统计上网者分布在每个网页的概率。

最初, 每个网页的概率均等, 每次跳转时, 网页 X 将其  $PR(\text{PageRank})$  均分到所指向的所有页面, 记链接数为  $L(X)$ , 于是, 经过一次跳转后:

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \dots$$

我们将每个网页抽象成一个节点, 超链接抽象为有向边, 共同构成一个图。则每次跳转可视为所有页面 PR 构成的特征向量  $R$  与该图的出度邻接矩阵  $M$  相乘, 即:

$$R = \begin{bmatrix} PR(p_1) \\ PR(p_2) \\ \vdots \\ PR(p_n) \end{bmatrix} \quad M = \begin{bmatrix} p_1 \rightarrow p_1 & p_2 \rightarrow p_1 & \cdots & p_n \rightarrow p_1 \\ p_1 \rightarrow p_2 & p_2 \rightarrow p_2 & \cdots & p_n \rightarrow p_2 \\ \vdots & \vdots & \ddots & \vdots \\ p_1 \rightarrow p_n & p_2 \rightarrow p_n & \cdots & p_n \rightarrow p_n \end{bmatrix}$$

$$R_1 = MR_0$$

多次迭代后, PR 值趋于稳定, 即为最终的 PR 值。

#### 2.4.2 设计思路

任务四的输入为任务三的输出, 格式如下:

人物 [名字<sub>1</sub>, 影响<sub>1</sub> | 名字<sub>2</sub>, 影响<sub>2</sub> | ... | 名字<sub>n</sub>, 影响<sub>n</sub>]

影响<sub>i</sub> 为名字<sub>i</sub> 与该人物归一化后的同现次数, 表示名字<sub>i</sub> 对该人物的影响权重。

每个人物视为图的一个节点, 边权重为二人同现次数。

对于普通的 PageRank 计算, 由于会存在自环边以及无出度的节点, 为方式到达某一节点后陷入该点, 会加入“随机浏览者”(random surfer)的概念, 即到达某个节点后有一定概率直接跳转到任意一个节点, 从而避免此情况。然而在此次任务中, 首先没有自身与自身同现的情况, 因此无自旋边; 同时 A 与 B 同现, 则 B 一定与 A 也同现, 因此不考虑权重时所有边实际都为无向边, 因此不存在出度为 0 的节点。所以此次任务无需引入“随机浏览者”。

MapReduce 框架下, 运算分布进行, 因此不使用邻接矩阵, 而采用邻接表的形式。算法大致分为三个阶段:

阶段一: 预处理首先要将输入格式化为供之后迭代处理的形式。采用如下格式: key: 人物 value: PageRank#[名字<sub>1</sub>, 影响<sub>1</sub> | 名字<sub>2</sub>, 影响<sub>2</sub> | ... | 名字<sub>n</sub>, 影响<sub>n</sub>]

以概率为初始值, PageRank 应设置为  $1/N$ , 但  $N$  值较大, 较小数字做乘法时误差较大, 因此将初始 PR 设置为 1 来减小误差。

阶段二：迭代计算迭代计算 PR 值，直到 PR 收敛。

在 Mapper 中，首先输出如下键值对：key：人物

value：# 出度表

此对目的在于维护出度表，value 前加 # 使 reducer 便于区分。

之后计算 PR 值，记 A 的出度表集合为 N, 则计算过程如下式：

$$NewPR(A) = \sum_{x \in N} OldPR(x) * weight(x \rightarrow A)$$

计算得到新的 PR 值，再输出一组键值对：key：人物

value：新 PageRank 值

在 Reducer 中，首先查看 value 前是否有 # 号以区分该键值对类型。由于是一个迭代过程，将输出格式化，与 Mapper 的输入格式相同。

阶段三：处理结果在 Mapper 阶段去除结果中的出度表，只保留 PR 值。

利用 Partition 类进行排序，由于默认为升序，结果需要降序，因此重写 DoubleWritable。因为只有 1000 余数据，未采用采样排序，使用了简单的全排序。

Reducer 阶段整理输出即可

### 2.4.3 代码讲解

程序可分为三个模块：PageRank、RageResultSort 以及调度模块。模块一：PageRank 此模块包含了阶段一与阶段二。

```
class PageRankMapper extends Mapper<Text, Text, Text, Text> {
    private Text resKey = new Text(); // 要传入reduce的key和value
    private Text resValue = new Text();

    @Override
    protected void map(Text key, Text value, Context context) throws IOException, InterruptedException {
        String name = key.toString();
        Double rank = 0.0;
        String relations = new String();
        if (value.toString().contains("#")) { // 不是第一次迭代
            rank = Double.parseDouble(value.toString().split("#")[0]); // 原本rank
            relations = value.toString().split("#")[1]; // 出度表 格式为[名字,影响|名字,影响|...]其中影响为对key中人物的影响
        } else { // 第一次迭代, 设置默认rank
            rank = 1.0; // 初始化rank值为1
            relations = value.toString(); // 出度表
        }
        resKey.set(name);
        resValue.set("#" + relations);
        context.write(resKey, resValue); // key为名字 value以#开头, 后面为出度表, 生成此键值对

        String[] slist = relations.split("\\[\\|\\]") [1].split("\\|"); // 分离出度表
        String target = new String();
        Double weight = 0.0;
        Double res = 0.0;
        for (String s : slist) {
            target = s.split(",")[0]; // 贡献影响的人名
            weight = Double.parseDouble(s.split(",")[1]); // 影响值作为权重
            resKey.set(target);
            res = weight * rank; // 得到此人对其造成的影响值
            resValue.set(res.toString());
            context.write(resKey, resValue); // 发送键值对, key为受影响的人物名字, value为造成的影响值
        }
    }
}
```

图 1. PageRankMapper

## 2.5 任务五

todo

## 2.6 任务六

todo

# 3 优化与改进

## 3.1 任务一

todo

## 3.2 任务二

todo

## 3.3 任务三

todo

## 3.4 任务四

原本迭代次数为 20 次，在检查中间结果时发现在 14 次之后，结果变化不大，基本收敛，因此将迭代次数改为 15 次，节省开销。

## 3.5 任务五

todo

## 3.6 任务六

todo

# 4 实验经验总结与改进方向

1) todo

2) todo