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、闫旭芃

2020 年 7 月 30 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，任务一、任务六，程序试运行与组织debug研讨
171860550, 王一之：算法设计与实验规划，任务四，参与debug，实验版本控制
171860549, 闫旭芃：算法设计与实验规划，任务五，参与debug与数据核对
171840565, 李展烁：算法设计与实验规划，任务一优化、任务二、任务三，参与debug并提出重要优化思路

1.2 任务要求

1.3 设计思路

2 实验实现

2.1 任务一：数据预处理

2.1.1 设计思路

数据输入：已经分词、分段的多篇中文文本文件

数据输出：每一段或几段中包含的所有人名，按顺序依次输出。

需要注意的是，由于我们的实验是对金庸全部人物关系的分析，因此无需特别地注意不同文件及文件名。

我们使用提供的名单列表，通过HashSet结构，快速地比对词语是否位于名单列表当中。

在装载名单列表时，由于姓名数量的有限性，且由于 configuration只能传输字符串，因此使用一个私有的将其全部装载到一个字符串中，填入configuration中，并在map的setup阶段将其提取出来，并存储于HashSet中。

特别地，为了提高效率，注意使用StringBuilder和String的相互配合。

2.1.2 程序分析

Main 主函数类：

为了传递名单列表，调用NameLoader类的load方法，从输入的文件名中获取名单，并使用configuration进行配置以备后用。其余大多为路径、类名的配置，不赘述。如图1

```java
public class GetCharaName {
    public static void main(String []args) throws Exception {
        Configuration myCfig = new Configuration(); // 配置初始化
        String[] argRemain = new GenericOptionsParser(myCfig, args).getRemainingArgs(); // 参数获取
        if (argRemain.length != 2 && argRemain.length != 3) {// 判断参数，错误判断，获取文件位置
            System.err.println("Usage: GetCharaName <in> <out> <<names>>");
            System.exit(2);
        }
        String pathNmFile = "../data/people_name_list.txt";// 给一个默认的文件路径
        if (argRemain.length == 3) {
            pathNmFile = argRemain[2];
        }
        NameLoader loader = new NameLoader();// 使用名单装载器装入名单
        String strAllName = loader.load(pathNmFile);

        Job job = Job.getInstance(myCfig, "GetCharaName");// 开始job，进行一系列设置
        job.setJarByClass(GetCharaName.class);
        job.setMapperClass(GetNameMapper.class);
        job.setReducerClass(GetNameReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);

        job.getConfiguration().set("Names", strAllName);// 使用configuration进行名单配置

        FileInputFormat.addInputPath(job, new Path(argRemain[0]));// 配置输入输出路径
        FileOutputFormat.setOutputPath(job, new Path(argRemain[1]));

        System.exit(job.waitForCompletion(true)? 0:1);

    }
}
```

图 1. 主函数类具体实现

NameLoader类：

使用FileInputStream获取文件内容，使用BufferReader进行缓冲区文件读入操作。

使用StringBuilder快速从单行读入中获取姓名，并快速构建字符串。如图2

```
class NameLoader {
    public String load(String filename) throws IOException {
        FileInputStream fIS = new FileInputStream(filename);// 使用文件名初始化文件输入流
        BufferedReader bR = new BufferedReader(new InputStreamReader(fIS));// 使用缓冲区读入文件流

        String ll = "";
        StringBuilder strAllName = new StringBuilder();// 使用StringBuilder快速构建名单
        while((ll = bR.readLine()) != null) {// 获取单行不为空
            strAllName.append(ll);
            strAllName.append(" ");
        }

        fIS.close();
        bR.close();

        return strAllName.toString();// 将构建好的名单字符串转成字符串格式
    }
}
```

图 2. 名单装载器具体实现

Mapper类：

在setup中从configuration中获取之前装载的名单字符串，经过分词处理后，存入HashSet的名单表中。

在map的重载函数中，对每行也即每个value值以空格为分隔符分割，并依此使用HashSet的查找操作比对是否属于姓名。并将姓名归并为一个字符串输出。如图3

Reducer类：

不做操作，直接将获取的值输出即可。

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

```java
class GetNameMapper extends Mapper<Object, Text, Text, NullWritable> {
    private HashSet<String> name_set = new HashSet<>();// 使用HashSet存储名单，以便快速比对单词与姓名
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        String names = context.getConfiguration().get("Names");
        // 在setup中从configuration中获取名单字符串
        for (String name : names.split(" ")) {
            name_set.add(name);
        }
        // 使用空格分词后依次存入HashSet
    }

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringBuilder sb = new StringBuilder();// 使用stringBuilder快速对单行有效姓名进行连接
        for (String term : value.toString().split(" ")) {
            if (name_set.contains(term)) {
                sb.append(term);
                sb.append(" ");
            }
        }
        // 使用空格分词，依次比对查找以后将有效姓名append到StringBuilder上

        if (sb.length() > 0) {
            context.write(new Text(sb.toString()), NullWritable.get());
        }
        // 把构建好的单行姓名列表输出
    }
}
```

图 3. mapper具体实现

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(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：

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

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与该图的出度表矩阵相乘，即：

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

$$R_1 = MR_0$$

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$_1$,影响$_1$—名字$_2$,影响$_2$—...—名字$_n$,影响$_n$]
影响$_i$为名字$_i$与该人物名字归一化后的同现次数，表示名字$_i$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

2.5 任务五

todo

2.6 任务六：基于PageRank的可视化

2.6.1 设计思路

金庸的全部武侠小说中总共包含人物一千余名，其存在的相互关联可能达到100K的等级，在这样庞杂的人物关系中，想要让全部的人物和关系呈现在受众面前，是不现实的。这一判断得到了很多前车之鉴的证实，例如使用gephi软件生成的关系图，如同一团乱麻，几乎看不清任何的姓名或关系，更枉谈“得到一些有趣的结论”。

在这样的前提下，我们选择了对可视化的内容进行取舍，将可视化的人物数量级从1000降低到了100的量级。同时对人物的重要性、人物关系的亲密度以PageRank及其排序的结果进行刻画，从而得到了比较好的视觉效果。

为了进行个性化的开发，我们选择了Qt作为可视化部分的开发平台，使用C++作为开发语言，编写了一个RelationPainter的程序，对数据进行个性化的、延拓性强的可视化操作。利用该程序，我们可以

- 通过文件装载按钮，使用txt文件作为输入，直接获得效果图；
- 通过关系曲线的粗细更直观地表示了人物关系的亲密程度；
- 通过人物姓名和标识点的尺寸，可以更直观地感受人物的重要性；
- 通过关系曲线的颜色，更清晰地辨别同一个人所具有的人物关系；
- 通过关系曲线的高亮，更快地辨别某一特定个体的关系网；
- 通过操控“显示最重要的n位人物”拖动条，控制显示在视野中的人数；
- 通过操控“调整显示尺寸”拖动条，控制人物关系显示的疏密程度，从而避免人物的重叠；
- 通过将人物名称和人物标识点围成圆环状，避免了姓名与人物关系线条的重叠。

该程序的上限很高，延拓性很广，我们将在后面相关章节中详细介绍其改进和优化思路。

2.6.2 程序分析

Relation 类：

这是一个刻画单个人物及与之相关的人物关系的类。其中，通过QList¡QPair¡QString,double¿¿维护了一个人物关系表，从而可以对于该人物相关的人物关系进行操作。如图4

```
// 这是一个刻画单行人物关系的类
class Relation
{
public:
    Relation(QString n, double r);
    // 添加新的关系
    void addNewRe(QString n, double w);

    //获取函数及简单的参数判断函数
    QString getName(){return name;}
    double getRank(){return rank;}
    QList<QPair<QString,double>*> getReList(){return reList;}

    int getListLen(){return reList.length();}
    bool isEmpty(){return reList.isEmpty();}
private:
    QString name; // 人物姓名
    double rank;// 人物的pageRank
    QList<QPair<QString,double>*> reList; // 与其他人的关系表
};
```

图 4. 单人物关系类具体实现

sigFig 类：

这是一个以单人物关系为基础的单图元类。在这个类中，维护了一个Relation类对象以存储人物关系。同时根据其中图元绘制参数常量的值，在绘制之前更新其图元的各项属性（如粗细、颜色、大小、角度、位置、透明度等），之后通过其中的绘制函数分别绘制人物标志点、人物名和人物关系曲线。一般来说，人物的PageRank值越高，人物名和标志点绘制得就越大越粗，而人物关系的权重越高，人物关系曲线就绘制得越粗。

特别的，由于我们将人物标志点和人物名绘制成圆盘状以避免重叠，而每个人物又在圆盘上占据不同的角度值，因此需要给每个人物图元在绘制前计算其在圆盘所占的角度。又因为每个单图元类没有其他图元的角度信息，因此在绘制关系曲线的时候，需要在更高层的类计算好角度对应表，并传入关系曲线的绘制函数。

由于类中大部分的方法为更新参数、获取或设置变量的方法，故不做展示，只集中展示genAll（更新所有参数），paintDot（绘制标志点），和paintLine（绘制标志曲线）三个函数的实现。

在图5中，我们看到，在对r（圆盘半径），midX、midY（屏幕中心X、y），mxRk（最大的PageRank值）设置过后，分别对CirWid（标志点粗细）、TxtWid（人物名粗细）、 DotR

```
// 汇集所有的生成函数
void sigFig::genAll(int r, int txtOff, double lasDeg, int lasR, double mxRk, int midX, int midY)
{
    setRXY(r,midX,midY);
    setMaxRank(mxRk);
    genCirWid();
    genTxtWid();
    genDotR();
    genDeg(r, lasDeg, lasR);
    genDotXY(r, midX, midY);
    genTxtXY(r + txtOff, midX, midY);
}
```

图 5. 单图元，全参数更新函数具体实现

（标志点大小）、Deg（标志点角度）、DotXY（标志点坐标）、 TxtXY（人物名坐标）进行了更新。 在图6可以看到，在更新了全部的绘制参数后，使用Qpainter、QPen、QFont等变量对画笔进行了设置，将画笔移动一定的角度和坐标，并依此绘制圆和Text文本。 在图7中，首先判断是否需要设置高亮色，并根据之前更新的参数设置画笔，此时注意，QPen的颜色设置内多了一个透明度，由于我们是通过绘制点集的方式绘制的曲线，因此透明度需要设置得足够低才有效果（当透明度低时，可以略去很多无效的曲线关系信息）。特别注意到，传入的degList是一个其他图元的角度表，通过对此表的查询，才能正确获得关系曲线的目标点位置，从而使用我们自己实现的简单贝塞尔曲线正确生成关系曲线。同时注意到，图元本身角度和目标点角度的比较，可以确认目标点与本身的排名先后，与限制位的比较，可以判断本身及目标点是否处于需要绘制的点集内，从而判断是否需要绘制曲线。

sigFigList 类：

作为一个统筹所有单图元的类，其主要的行为是对单图元的参数在宏观上进行调控，例如控制显示个数、疏密程度、分发颜色、更新角度表等操作。如图8

mainwindow 类：

对画布上产生的各类信号给出对应的槽进行处理，具体的任务交给 fgLs去做。如载入文件、设置显示个数以及疏密程度的滚动条、鼠标移动设置高亮的判断等。如图9

2.6.3 结果展示

打开程序，导入文件并调整滚动条至合适位置，可以看到最重要的n位人物及其关系列表出现在屏幕中央。几条最粗的曲线彰显了郭靖与黄蓉、杨过与小龙女、胡斐与程灵素、慕容复与王语嫣、张无忌与周芷若等等人物之间的密切关系。通过不同的颜色，可以相对容易地查找与同一人物相关的人物关系。

3 实验经验总结与改进方向

1) todo

2) todo

3) todo

4) todo

```
// 画点和人名
void sigFig::paintDot(QMainWindow *q)
{
    QPainter pt(q);
//    QPen pen(Qt::darkBlue);

    if (highlight) lineColor = hlColor;
    QPen pen(lineColor);
//    if (highlight) pen.setColor(hlColor);

// 根据计算好的位置和角度，旋转画笔并绘图即可——此时的颜色不需要设置透明度

    pt.save();
    pt.translate(xDot, yDot);
    pen.setWidth(cirWid);
    pt.setPen(pen);
    pt.rotate(getDeg180()-45);
    pt.drawEllipse(0,0,dotR,dotR);
    pt.restore();

    pt.save();
    pt.translate(xTxt, yTxt);
    pt.rotate(getDeg180());
//    pen.setWidth(txtWid);
    QFont font("黑体", txtWid, QFont::Bold, false);
    pt.setFont(font);
    pt.setPen(pen);
    pt.drawText(0,0,rel->getName());
    pt.restore();
}
```

图 6. 单图元，绘制标志点函数具体实现

```
// 绘制曲线的函数
// 需要先访问map的角度字典，然后找到目标点的位置
// 以起点、圆心和目标点为基点，下压参数作为调整值，绘制贝塞尔曲线
// 此时的绘制时需要设置透明度，不然结果就没法看了
void sigFig::paintLine(QMainWindow *q, std::map<QString, double> degList, QString limitName)
{
    QPainter pt(q);
//    QPen pen(QColor(32,32,160,lineTransp));
//    QPen pen(lineColor, lineTransp);
    if (highlight) lineColor = hlColor;
    QPen pen(QColor(lineColor.red(),lineColor.green(), lineColor.blue(), lineTransp));
    for (QPair<QString,double>* qp: rel->getReList())
    {
        auto iter = degList.find(qp->first);
//        int limitPos =  min(degList.size(),vitalNum) - 1;
        double limitDeg = degList.find(limitName)->second;
        if (iter != degList.end() && iter->second > deg && iter->second <= limitDeg)
        {
            QPair<int,int> resXY = calDotXY(iter->second);
            QPoint a(xDot,yDot);
            QPoint b(xMid,yMid);
            QPoint c(resXY.first,resXY.second);
            b = (1-lowSize)*(a+c)/2.0 + lowSize*b;
            QList<QPoint> pts = genBesLine(a,b,c);

            pt.save();
            int lineWid = (baseLineWid + maxLineOff * qp->second/standardWei);
            pen.setWidth(lineWid);
            pt.setPen(pen);
            for(QPoint qpt:pts)
            {
                pt.drawPoint(qpt);
            }
            pt.restore();
        }
    }

}
```

图 7. 单图元，绘制关系曲线函数具体实现

```cpp
// 原本想搞个随机化，想了想觉得没必要
// 向每个sigFig分发颜色
void sigFigList::genColor()
{
    for(int i = 0; i< sigLs.length();++i)
    {
        sigLs[i]->setLineColor(allColor[i%allColor.length()]);
    }
}

void sigFigList::genMaxRank()
{
    maxRank = sigLs[0]->getRel()->getRank();
}

// 这里如果不清空则会造成目标点无法调整的情况，因为map的性质使然
void sigFigList::genDegList()
{
//    if (!sigLs.isEmpty())
//        sigLs.clear();
    degList.clear();
    for(sigFig * ss: sigLs)
    {
        degList.insert(std::map<QString, double>::value_type(ss->getRel()->getName(), ss->getDeg()));
    }
}

// 必须依次访问sigFig成员并计算角度等参数，才能正常地全部更新
void sigFigList::genAll()
{
    genColor();
    double curDeg = 0;
    int curDotR = 0;

    genMaxRank();
    for(sigFig * s: sigLs)
    {
        s->setSigSize(sigSize);
        s->genAll(cirR, txtOff, curDeg, curDotR, maxRank, midX, midY);
        curDeg = s->getDeg();
        curDotR = s->getDotR();
    }
    genDegList();
}
```

图 8. 分发颜色、更新角度列表、更新疏密程度具体实现

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    void loadFile(); // 装入文件
//    void paintRe();

protected:

void paintEvent(QPaintEvent *); // 绘制图像

void mouseMoveEvent(QMouseEvent *event); // 捕捉鼠标事件

private slots:

    void on_FileBrowse_clicked();// 对装载文件的按钮的处理

    void on_vitalNumSlider_valueChanged(int value);// 对显示数量滚动条变化的处理

    void on_sigSizeSlider_valueChanged(int value);// 对疏密程度滚动条变化的处理

    void myMouseMoveHandler(QMouseEvent * e);// 对鼠标点击事件的处理

signals:
    void mouseMove(QMouseEvent *event);// 发射的鼠标信号

private:
    Ui::MainWindow *ui;
//    QList<Relation*> reLs;
    sigFigList fgLs;// 包含一个sigFigList类的成员，以对关系和显示进行处理
    bool loaded;// 已经装载文件的信号，应当放入sigFigList更好

    const int hlDistance = 10;//控制高亮显示的检测半径，也即灵敏度

};
```

图 9. mainwindow具体实现

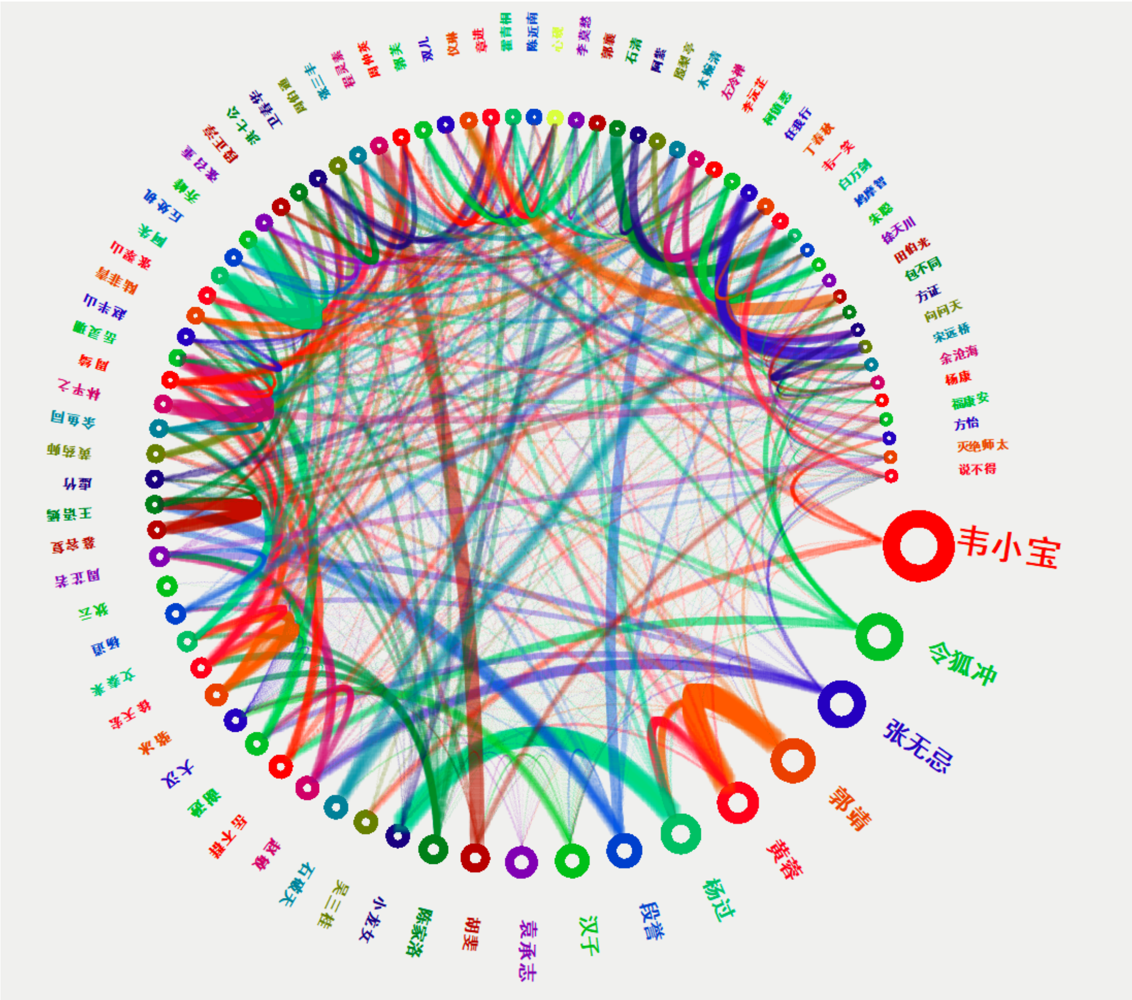

图 10. 可视化结果

5) todo

6) todo

7) 在任务六中，由于QPainter的绘制特性缘故，因此人物的姓名有一半是倒着的，造成观感不佳，需要解决。

8) 在任务六中，由于曲线使用点集绘制，而单点的绘制被自动设置为方形，因此曲线的形状不佳，需要改进。

9) 在任务六中，由于曲线使用的颜色的亮度、深浅不同，造成人物关系密切程度的直观性降低，需要改进颜色组。

10) 在任务六中，采用数据结构效率较低，对于冗余的判断和循环没有进行优化，造成操作上的延迟，需要优化。

11) 在任务六中，由于采用的是高Rank值的人物向低Rank值的人物绘制曲线，因此造成低rank值人物快速无法快速辨认全部与其相关的人物关系。可以采用颜色的渐变进行优化。

12) 在任务六中，尚存在很多绘制参数没有在外部留出接口，例如透明度无法在用户界面设置，需要改进。