一.实验原理

实验基于简单共现关系，编写 Python 代码从纯文本中提取出人物关系网络，并用Gephi 将生成的网络可视化。
实体间的共现是一种基于统计的信息提取。关系紧密的人物往往会在文本中多段内同时出现，可以通过识别文本中已确定的实体（人名），计算不同实体共同出现的次数和比率。当比率大于某一阈值，我们认为两个实体间存在某种联系。

二.环境搭建

打开Xfce终端，进入 /home/shiyanlou/Code 目录，创建 work 文件夹, 将其作为课程的工作目录。下载并安装 gephi 。

```
$ cd /home/shiyanlou/Code
$ mkdir work && cd work
$ mkdir gephi && cd gephi
$ wget http://labfile.oss.aliyuncs.com/courses/677/gephi-0.9.1-linux.tar.gz
#下载
$ tar -zxvf gephi-0.9.1-linux.tar.gz
#解压
```

在 /home/shiyanlou/Code/work/ 目录下载《釜山行》的中文剧本。

```
$ wget http://labfile.oss.aliyuncs.com/courses/677/busan.txt
```

安装jieba中文分词。

```
$ sudo pip3 install jieba
```

三.实验步骤

1. 分析：对于简单网络，建立词典进行实体识别是更有效的方式
   可以通过各类百科获取《釜山行》的主要人物，你可以在百度百科中找到他们的介绍，并将人名写入一个字典中。项目将主要人物的名称保存在文件dict.txt中，你可以通过下面的命令下载这个字典，也可以自己新建一个文件保存。字典dict.txt需放在文件夹/home/shiyanlou/Code/work下。

```
$ wget http://labfile.oss.aliyuncs.com/courses/677/dict.txt
```

2. 确定变量
   在/home/shiyanlou/Code/work 目录下创建代码文件busan.py,开始进行python代码的编写。

在代码中，使用字典类型names保存人物，该字典的键为人物名称，值为该人物在全文中出现的次数。使用字典类型relationships保存人物关系的有向边，该字典的键为有向边的起点，值为一个字典edge，edge的键是有向边的终点，值是有向边的权值，代表两个人物之间联系的紧密程度。
lineNames是一个缓存变量，保存对每一段分词得到当前段中出现的人物名称，lineName[i]是一个列表，列表中存储第i段中出现过的人物。

```
# -*-  coding: utf-8 -*-
import os, sys
import jieba, codecs, math
import jieba.posseg as pseg

names = {}            # 姓名字典
relationships = {}    # 关系字典
lineNames = []        # 每段内人物关系
```

3. 计算人物出现次数
   在具体实现过程中，读入《釜山行》剧本的每一行，对其做分词（判断该词的词性是不是“人名”[词性编码：nr]，如果该词的词性不为nr，则认为该词不是人名），提取该行（段）中出现的人物集，存入lineName中。之后对出现的人物，更新他们在names中的出现次数。

```
jieba.load_userdict("dict.txt")        # 加载字典
# 加载自定义词库，每一行包括一个词，词频，词性，词频可省略
# codecs保证以特定的编码形式打开文件
with codecs.open("busan.txt", "r", "utf8") as f:
    for line in f.readlines():
        poss = pseg.cut(line)         # 分词并返回该词词性
        lineNames.append([])          # 为新读入的一段添加人物名称列表
        for w in poss:
            if w.flag != "nr" or len(w.word) < 2:
                continue              # 当分词长度小于2或该词词性不为nr时认为该词不为人名
# 注意这种用法，直接定位视在操作的行，不需要额外定义变量
            lineNames[-1].append(w.word)         # 为当前段的环境增加一个人物
            if names.get(w.word) is None:
                names[w.word] = 0
                relationships[w.word] = {}
            names[w.word] += 1                   # 该人物出现次数加 1
```

你可以在 with 代码块之后添加以下代码输出生成的 names 来观察人物出现的次数：

```
for name, times in names.items():
    print(name, times)
```

4. 根据识别结果构建网络
   对于 lineNames 中每一行，我们为该行中出现的所有人物两两相连。如果两个人物之间尚未有边建立，则将新建的边权值设为 1，否则将已存在的边的权值加 1。这种方法将产生很多的冗余边，这些冗余边将在最后处理。

```
for line in lineNames:                    # 对于每一段
    for name1 in line:
        for name2 in line:                # 每段中的任意两个人
            if name1 == name2:
                continue
            if relationships[name1].get(name2) is None:        # 若两人尚未同时出现则新建项
                relationships[name1][name2]= 1
            else:
                relationships[name1][name2] = relationships[name1][name2]+ 1         # 两人共同出现次数加 1
```

5. 过滤冗余边并输出结果
   将已经建好的 names 和 relationships 输出到文本，以方便 gephi 可视化处理。输出边的过程中可以过滤可能是冗余的边，这里假设共同出现次数少于 3 次的是冗余边，则在输出时跳过这样的边。输出的节点集合保存为 busan_node.txt ，边集合保存为 busan_edge.txt 。

```
with codecs.open("busan_node.txt", "w", "gbk") as f:
    f.write("Id Label Weight\r\n")
    for name, times in names.items():
        f.write(name + " " + name + " " + str(times) + "\r\n")

with codecs.open("busan_edge.txt", "w", "gbk") as f:
    f.write("Source Target Weight\r\n")
    for name, edges in relationships.items():
        for v, w in edges.items():
            if w > 3:
                f.write(name + " " + v + " " + str(w) + "\r\n")
```

完成所有代码编写后，运行
在文件夹work 下将会生成busan_node.txt 和busan_edge.txt 。
6. 可视化网络