Step中的数据格式，是由2个shopID组成，比如1201，就是商店12传到商店01发生的消息传递事件

1.getTwo1(String s)函数（2个聚集点之间）

参数：

FileName:读取文件的文件名。

First：是否是第一次取，因为后面代码会涉及和上一步的比较，所以第一次读取数据是需要布尔值来判断。

Sid[]:每个商店里的人数（通过计算人的离开来计算人的数目）。

St[]：存储上一步人携带的商店信息。

Sum:总的消息传递事件（1000步）。

Max:每个商店里最大的人数（1000步）。

Ave：总的一次实验中（1000步）在聚集点事件中的平均人数。

Ns:前提是聚集点消息传递事件发生。在上一步，在这2个发生消息传递事件的商店里的总人数。

Na:前提是聚集点消息传递事件发生。在这一步，在这2个发生消息传递事件的商店里的总人数。

返回：

Int[]{sum,ave,max}

2.getThree1(String s)函数（3个聚集点之间）

大部分参数与getTwo1相同，不同部分：

在475行判断是否为事件的时候，加了一个条件，若重复则跳出循环，在判断3个聚集点是否有消息传递时候，这个很好理解。（比如我从商店1到商店2，又回到商店1，这就不是3个聚集点的事件）

返回：

Int sum

3.getTwoAll1（String fileName,int n）函数

读取所有的step文档，进行总的n次实验的取平均值

返回：

Int[]{sum,ave,max}（均为平均值）

4.getThreeAll1（String filename,int n）函数

读取所有的step文档，进行总的n次实验发生事件总和的取平均值

返回：

Int sum

5.dataCompare函数

参数：

Fileshop,brs,s是用来读取step.txt文件

Cmp 同步和异步之间的比较

Maxpass最长链的长度

Three三次消息传递事件发生的总数

Two俩次消息传递事件发生的总数

Del 除去小概率下发生的消息传递事件

Ss 存储每行事件的个数

Pass 俩次消息传递链的长度（像1到2,2到3，链的长度就相当于2），每次操作完后再次初始化为1。

Max记录传递链的最大长度

Pass1 三次消息传递链的长度，每次操作完后再次初始化为1.

Pass2 所有消息传递链的平均长度

Avepass 记录消息链的长度

Event 存储step文件中每行的消息传递事件的个数.

En 计算消息传递事件发生的总的个数

listofTwo：包含所有二次消息事件传递的信息，以2个shopID形式存储一个事件

listofThree：包含所有三次消息事件传递的信息，以3个shopID形式存储一个事件

关键代码说明：

**for** (**int** i = 0; i < ss.length; i++) {

**if** (ss.length > 1)

**if** (!listoftwo.contains(Integer.*parseInt*(ss[i])))

listoftwo.add(Integer.*parseInt*(ss[i]));

**int** k = i;

**for** (**int** j = i + 1; j < ss.length; j++) {

这里是取模得到后2位的商店ID和除以100得到前2位商店ID，判断是否为消息传递链

n = Integer.*parseInt*(ss[k]) % 100;

m = Integer.*parseInt*(ss[j]) / 100;

**if** (n == m) {

System.*out*.println(nn + "pass:" + ss[k] + " "

+ ss[j]);

k = j;

pass++;

}

}

**if** (pass > max) {

max = pass;

}

pass = 1;

}

// System.out.println(nn+"max:"+max);

maxpass += max;

以上部分代码将明显的2个消息事件传递全部存入listoftwo，同时得到最大消息传递链长度，用max存储，同时pass重置为1。

**for** (**int** i = 2; i <= max; i++) {

**int** flag = 0;

**for** (**int** k = 0; k < ss.length; k++) {

**int** h = k, h1 = k;

**for** (**int** j = k + 1; j < ss.length; j++) {

n = Integer.*parseInt*(ss[h]) % 100;

m = Integer.*parseInt*(ss[j]) / 100;

**if** (n == m) {

// System.out.print(ss[h]+ss[j]+" ")

**if** (pass1 == 1)

flag = j;

h = j;

pass1++;

}

对如2个，3个，4个链长度进行潜在辨别的时候，pass1在超过或等于我们原先设置的i（链长度）时候，对如（0102,0203）辨别为（0103）为一个2个商店的消息传递，3个商店的潜在消息传递同理。其中flag相当于中间者的标志符。

**if** (pass1 >= i) {

avepass.add(pass1);

**int** ntwo = (Integer.*parseInt*(ss[h1]) / 100 + 0)

\* 100 + Integer.*parseInt*(ss[j]) % 100;

// System.out.print("a"+ss[h]+ss[j]+" "+ntwo);

**int** nthree = Integer.*parseInt*(ss[h1]) \* 100

+ Integer.*parseInt*(ss[j]) % 100;

**if** (!listoftwo.contains(ntwo))

listoftwo.add(ntwo);

**if** (!listofthree.contains(nthree))

listofthree.add(nthree);

j = flag;

h = h1;

pass1 = 1;

}

}

pass1 = 1;

}

}

这段代码对潜在的2个商店消息传递和3个商店的消息传递进行辨别并进行存储