

数据结构实验报告



实验课程： 数据结构

实验名称： 学术文本中词的共现网络构建实验

学生姓名： 夏雪

学 号： 2014302330007

指导教师： 陆泉

目 录

一、实验目的.....	2
二、实验内容.....	2
三、实验原理.....	2
四、实验步骤.....	2
实验阶段一：文本分词并统计词频	3
实验阶段二：建立摘要-词稀疏矩阵	5
实验阶段三：建立共现网络	8
实验阶段四：剔除虚词并输出	9
五、实验结果.....	12

一、实验目的

熟悉数据结构与算法分析的基本理论，掌握各种数据结构的设计与使用，掌握栈、队列、串、数组、二叉树、图等数据结构和排序方法的基本运用，重点掌握在文本分析中综合运用上述知识与技能。

二、实验内容

选取“data mining”和“information architecture”作为两个关键词，从 Web of Science 学术数据库中检索选取 10 篇学术论文的英文摘要文本（每个关键词各五篇），进行包括下面各处理阶段的一系列文本处理：文本分词、统计词频、建立摘要-词的稀疏矩阵、分析词之间的共现关系并建立这 10 篇摘要文本的词共现网络。

三、实验原理

主要思路：

1. 选择 10 篇摘要，将他们保存下来。
2. 分别将每篇摘要的单词分离出来，统计各个单词在每篇摘要中出现的频率，并且按照字母顺序排序。
3. 将十篇摘要中的所有单词归并。
4. 建立十字链表，每行代表一篇摘要，每列代表一个单词，若结点不为空，结点值代表该列单词在该行摘要中出现的频次。
5. 建立共现网络。网络类型是无向网，每个结点代表一个单词，边的权值为端点所代表的词在摘要中共同出现的次数。
6. 按照共现此处从高到低的顺序输出共现的单词和共现次数，剔除无意义的词。

四、实验步骤

实验分为了文本分词并统计词频、建立摘要-词稀疏矩阵、建立共现网络、剔除虚词并输出这四大阶段。下面分阶段总结实验目的、实验基础、解决思路、关键代码、典

型实验数据与处理结果截屏、以及可能有的心得体会。

实验阶段一：文本分词并统计词频

1. 目的

把十篇摘要中的单词分出来并储存，同时统计词频。

2. 实验基础

无

3. 解决思路

- 1) 首先将十篇摘要新建为十个 `String` 字符串：`str1~str10`。
- 2) 创建一个顺序串 `SeqString` 的子类：`DvdStr`，改写 `SeqString` 的构造方法，新建数组 `DvdStr [] paperAbstract = new DvdStr [10]`用于保存摘要。
- 3) 新建一个 `Count` 类作为计数器，`Count` 类有两个数据域：`String term` 用于存放单词，`int freq` 用于存放词频。
- 4) 利用并拓展已有程序。`Exercise4_3_1`。改编课本第四章课后习题第三大题的第 1 题（该题统计字符串的单词个数而没有分词），在 `DvdStr` 类中编写成员函数 `Divide()`，该函数用于分词并统计当前字符串中的单词个数。在记录中查找，如无则插入（要排序），如有则累计。
- 5) 将十篇摘要分词并统计词频，结果存放在 `Divide[][]`二维数组中。

4. 关键代码

```
public Count[] Divide() { //方法：分词并计数
    Count[] result=new Count[2000];
    for(int i=0;i<2000;i++){
        result[i]=new Count();
    }

    int sum=0; //词数初始化为 0
    char currChar,preChar;

    String currWord=new String();

    currWord=currWord+this.charAt(0);
    for (int i = 1; i < this.length(); i++) {
```

是字母

```
currChar = this.charAt(i);    //当前字符
preChar = this.charAt(i - 1); //前一个字符
if (((int) (currChar) < 65 || (int) (currChar) > 122 //当前字符不是字母
    || ((int) (currChar) > 90 && (int) (currChar) < 97))
    && (((int) (preChar) >= 65 && (int) (preChar) <= 90) //当前字符的前一个字符
        || ((int) (preChar) >= 97 && (int) (preChar) <= 122)))
{
    if(sum==0){
        result[sum]=new Count(currWord);
        sum++;
    }
    else{
        int j=0;
        for(j=0;j<sum;j++){
            if (currWord.equals(result[j].getTerm())){//如果有相同的词，词频加 1
                result[j].setFreq((int) (result[j].getFreq()+1));
                j=-1;
                break;
            }
        }
        if(j!=-1){//没有相同的词，新建 result[]
            result[sum]=new Count(currWord);
            sum++;
        }
    }
    currWord=new String();
}

else{
    if((((int) (charAt(i)) >= 65 && (int) (charAt(i)) <= 90)
        || ((int) (charAt(i)) >= 97 && (int) (charAt(i)) <= 122)))
        currWord=currWord+this.charAt(i);
    }
}

for(int p=0;p<sum;p++){//单词排序
    for(int q=p;q<sum;q++){
        if (result[q].getTerm().compareTo(result[p].getTerm())<0){
            Count temp=new Count();
            temp.setTerm (result[p].getTerm());
            temp.setFreq (result[p].getFreq());
            result[p].setTerm(result[q].getTerm());
            result[p].setFreq(result[q].getFreq());
        }
    }
}
```

```

        result[q].setTerm(temp.getTerm());
        result[q].setFreq(temp.getFreq());
    }
}
return result;
}
}

```

5. 处理结果截屏

编写 testDivide 测试类 测试结果如下

```

原字符串为 shine bright like a diamond
shine bright like a diamond
shine bright like a diamond
find light in a beautiful sea I choose to be happy
单词I出现了1次
单词a出现了4次
单词be出现了1次
单词beautiful出现了1次
单词bright出现了3次
单词choose出现了1次
单词diamond出现了3次
单词find出现了1次
单词in出现了1次
单词light出现了1次
单词like出现了3次
单词sea出现了1次
单词shine出现了3次
单词to出现了1次
共有14个不同的词

```

6. 心得体会

此阶段最关键的部分是分词和统计词频的算法。课后习题通过判断单词的结束来统计单词总数,将这个算法做一些改动,每当判断一个单词结束,就将该单词保存在 result[] 数组中,并且同时统计词频:如果已经保存过该单词,将词频加一,否则新建一个数组元素。最后将 result[] 数组中所有的元素按照单词的首字母排序。

最初想法是:先判断单词开始,再判断单词结束,用截取子串的方法分词,后来发现这种方法将分词复杂化了而且没有提升算法性能,故最后采取只判断结束的方法。

实验阶段二：建立摘要-词稀疏矩阵

1. 目的

建立十字链表，每行代表一篇摘要，每列代表一个单词，若结点不为空，结点值代表该列单词在该行摘要中出现的频次。

2. 实验基础

已经得到每篇摘要的单词以及词频，保存在 Divide[][]数组中。

3. 解决思路

- 1) 将第二章中的 Node 和 LinkList 类分别改写为: MyNode 类和 MyLinkList 类。MyNode 的结点值为 String 类，MyLinkList 的结点类型为 MyNode。
- 2) 将十篇摘要中的所有词归并，按照字母顺序排序并保存在一个中，十篇摘要总单词数为 sum。
- 3) 建立 10*sum 十字链表，共 10 行，sum 列。
- 4) 第 i 行第 j 列的结点值表示第 j 个单词在第 i 篇摘要中出现的次数。

4. 关键代码

```
MyLinkList allWord=new MyLinkList();//归并十个摘要中所有不同的词
```

```
String Total=new String();
Total+=str1+str2+str3+str4+str5+str6+str7+str8+str9+str10;
DvdStr AllAbstract = new DvdStr(Total);
Count[] TotalCount=new Count[2000];
TotalCount= AllAbstract.Divide();
```

```
for(int i=0;i<2000;i++){
    if(TotalCount[i].getTerm()!=null&&TotalCount[i].getTerm()!="")
        allWord.insert(TotalCount[i].getTerm());
    else break;
}
int sum=allWord.length();
```

```
System.out.println("十篇摘要中共有"+sum+"个不同的词");
```

```
CrossList countWord=new CrossList(10,sum);//创建十字链表
```

```
for(int i=0;i<10;i++){
    for(int j=0;j<sum;j++){
        if(Divide[i][j].getTerm()!=null&&Divide[i][j].getTerm()!=""){
            int index=allWord.indexOf(Divide[i][j].getTerm());
```

```

        countWord.Insert(i+1,index+1, Divide[i][j].getFreq());//存放词频
    }
    else break;
}
}
String[] word=new String[sum];
for(int i=0;i<sum;i++){
    word[i]=TotalCount[i].getTerm();
}
countWord.printByTriple();//输出十字链表

```

5. 处理结果截屏

十篇摘要中共有**657**个不同的词
摘要-词的稀疏矩阵的三元组形式输出为

行 列 值

1	12	1
1	17	1
1	18	1
1	31	1
1	34	1
1	37	1
1	40	1
1	53	1
1	62	1
1	73	2
1	81	1
1	83	2
1	89	4
1	92	1
1	97	4
1	116	1
1	120	5
1	125	1
1	127	1
1	128	1
1	140	1

6. 心得体会

这一阶段个人认为有些地方处理不够恰当：在归并十篇摘要中所有单词的时候，选择的方法是将十篇摘要连接起来再用分词的方法，而不是利用前面分词的结果归并。

优点：与三元组顺序表相比，系数矩阵用十字链表表示，可以节省存储空间，输出结果简单清晰（不含值为零的结点）。

实验阶段三：建立共现网络

1. 目的

建立共现网络。网络类型是无向网，每个结点代表一个单词，边的权值代表端点所代表的词在摘要中共同出现的次数。

2. 实验基础

已经得到摘要-词的十字链表。

3. 解决思路

- 1) 建立共现网络，有 sum 个结点，结点值初始化为各个单词的字符串。
- 2) 储存边的权值，修改无向网的创建方法，权值初始值为 0。
- 3) 横向扫描十字链表。若第 i 行的第 j 和第 k 列结点值分别为 m 和 n ，则第 j 和第 k 个单词在第 i 篇摘要共现了 $\min\{m,n\}$ 次。
- 4) 将每一行中两个单词的共现次数相加，即得到这两个词在 10 篇文献中的共现次数，将以这两个词为端点的边赋权值为他们的共现次数。

4. 关键代码

```
MGraph collocate=new MGraph();//创建无向网 collocate  
collocate.createUDN(sum,sum*sum,word);
```

```
for(int i=0;i<10;i++){//给边赋权值，横向扫描
```

```
    OLNode rtemp1 =countWord.rhead[i];  
    rtemp1 = rtemp1.getRight;
```

```
    for (int j = 0; j < sum; j++) {
```

```
        if (rtemp1 != null && rtemp1.col == j+1) { //第 j+1 个词在第 i+1 行出现过  
            OLNode rtemp2 =rtemp1.getRight;  
            for(int k=j+1; k<sum; k++){
```

```
                if (rtemp2 != null && rtemp2.col == k + 1){ //第 k+1 个词在第 i+1 行出现过
```

```
                    collocate.arcs[j][k]+=min(rtemp1.e,rtemp2.e); //两个词的共现次数取较小值  
                    collocate.arcs[k][j]=collocate.arcs[j][k];
```

```

        rtemp2 = rtemp2.getRight();
    }
}
rtemp1 = rtemp1.getRight();
}

}
}

```

5. 处理结果截屏

无

6. 心得体会

此阶段最关键的部分是找出单词两两共现的次数，给共现网络的边赋权值。算法可读性高但是含有三层嵌套循环，扫描一篇摘要的共现网络时间复杂度是 $O(n^3)$ ，应该还有提升算法性能的空间。

实验阶段四：剔除虚词并输出

1. 目的

剔除虚词，输出共现网络。

2. 实验基础

已经得到词的共现网络。

3. 解决思路

- 1) 将文摘中高频出现的虚词（无意义的词）赋值给静态变量 X1~X21
- 2) 按照边的权值由高到低输出词的共现次数
- 3) 输出前判断两个词中是否有虚词，若有则不输出。

4. 关键代码

```

static final String X1="a"; //将文摘中高频出现的虚词赋值给静态变量 X1~X21
static final String X2="an";
static final String X3="and";
static final String X4="for";
static final String X5="in";
static final String X6="is";
static final String X7="are";
static final String X8="the";

```

```

static final String X9="to";
static final String X10="on";
static final String X11="that";
static final String X12="this";
static final String X13="of";
static final String X14="The";
static final String X15="as";
static final String X16="In";
static final String X17="we";
static final String X18="be";
static final String X19="by";
static final String X20="This";
static final String X21="with";

```

```

for(int k=60;k>3;k--) //按照边的权值由高到低输出词的共现次数
    for(int i=0;i<sum;i++){
        for(int j=i+1;j<sum;j++){
            if(collocate.arcs[i][j]==k)
                if(word[i].compareTo(X1)!=0&&word[i].compareTo(X2)!=0 //剔除虚词
                    &&word[i].compareTo(X3)!=0&&word[i].compareTo(X4)!=0
                    &&word[i].compareTo(X5)!=0&&word[i].compareTo(X6)!=0
                    &&word[i].compareTo(X7)!=0&&word[i].compareTo(X8)!=0
                    &&word[i].compareTo(X9)!=0&&word[i].compareTo(X10)!=0
                    &&word[i].compareTo(X11)!=0&&word[i].compareTo(X12)!=0
                    &&word[i].compareTo(X13)!=0&&word[i].compareTo(X14)!=0
                    &&word[i].compareTo(X15)!=0&&word[i].compareTo(X16)!=0
                    &&word[i].compareTo(X17)!=0&&word[i].compareTo(X18)!=0
                    &&word[i].compareTo(X19)!=0&&word[i].compareTo(X20)!=0
                    &&word[i].compareTo(X21)!=0
                    &&word[j].compareTo(X1)!=0&&word[j].compareTo(X2)!=0
                    &&word[j].compareTo(X3)!=0&&word[j].compareTo(X4)!=0
                    &&word[j].compareTo(X5)!=0&&word[j].compareTo(X6)!=0
                    &&word[j].compareTo(X7)!=0&&word[j].compareTo(X8)!=0
                    &&word[j].compareTo(X9)!=0&&word[j].compareTo(X10)!=0
                    &&word[j].compareTo(X11)!=0&&word[j].compareTo(X12)!=0
                    &&word[j].compareTo(X13)!=0&&word[j].compareTo(X14)!=0
                    &&word[j].compareTo(X15)!=0&&word[j].compareTo(X16)!=0
                    &&word[j].compareTo(X17)!=0&&word[j].compareTo(X18)!=0
                    &&word[j].compareTo(X19)!=0&&word[j].compareTo(X20)!=0
                    &&word[j].compareTo(X21)!=0)
                    System.out.println(" 单词    "+word[i]+" 和 单词    "+word[j]+" 共现
"+collocate.arcs[i][j]+"次");
            }
        }
    }

```

5. 处理结果截屏

十篇摘要中，按照单词共现次数由高到低，共现结果为：

单词 **big** 和单词 **data**共现**11**次
单词 **articles** 和单词 **news**共现**10**次
单词 **learning** 和单词 **performance**共现**10**次
单词 **data** 和单词 **mining**共现**9**次
单词 **based** 和单词 **big**共现**8**次
单词 **based** 和单词 **data**共现**8**次
单词 **big** 和单词 **mining**共现**8**次
单词 **data** 和单词 **social**共现**8**次
单词 **learners** 和单词 **learning**共现**8**次
单词 **learners** 和单词 **performance**共现**8**次
单词 **based** 和单词 **information**共现**7**次
单词 **based** 和单词 **mining**共现**7**次
单词 **different** 和单词 **information**共现**7**次
单词 **information** 和单词 **large**共现**7**次
单词 **information** 和单词 **learning**共现**7**次
单词 **information** 和单词 **system**共现**7**次
单词 **IA** 和单词 **information**共现**6**次
单词 **IA** 和单词 **organizations**共现**6**次

6. 心得体会

在将共现网络按照边权值由高到低的顺序输出的时候，首先考虑把边的权值排序，并且保留端点单词的序号。后来考虑到这种方法需要分配额外的储存空间，而且暂时没有想出一个高效率的算法，因此仅仅在输出前用循环语句控制边的权值，如果今后发现更高效的算法会再修改程序。

剔除无意义的词的时候，首先想到的是统计词语在前五篇摘要和后五篇摘要各自出现的次数，由于前后五篇摘要分别是不同的主题，所以，若一个单词在前后五篇中都高频出现，则判断它是虚词并且将它剔除。按照这种思路编程后的输出结果并不理想，因为难以对“高频出现”作出定量判断（出现多少次算高频？），并且发现误删了一些有意义的词如“information”，而且有些出现频率相对较低的虚词也并没有剔除出去。考虑到经常出现的虚词数量并不多，故这里将虚词列举出来并判断共现网络的结点是否是虚词，如果是虚词，则不输出。这种方法的优点是便于修改程序（新增虚词库），缺点是需要进行一系列的判断，算法较复杂。同样，如果今后发现更好的剔除虚词的方法，会加以改进。

五、实验结果

从 Web of Science 学术数据库中检索选取的 10 篇学术论文的英文摘要文本，进行文本分词、统计词频、建立摘要-词的稀疏矩阵、分析词之间的共现关系，并建立了这 10 篇摘要文本的词共现网络，最终可以按照三元组形式输出摘要-词稀疏矩阵，并且按照共现次数由大到小输出词的共现网络。

```
Console
<terminated> MidTerm [Java Application] C:\Program File
十篇摘要中共有657个不同的词
摘要-词的稀疏矩阵的三元组形式输出为
行 列 值
1    12    1
1    17    1
1    18    1
1    31    1
1    34    1
1    37    1
1    40    1
1    53    1
1    62    1
1    73    2
1    81    1
1    83    2
1    89    4
1    92    1
1    97    4
1   116    1
1   120    5
1   125    1
1   127    1
1   128    1
1   140    1
```

...

...

10 655 4

10 657 3

十篇摘要中，按照单词共现次数由高到低，共现结果为：

单词 **big** 和单词 **data**共现**11**次

单词 **articles** 和单词 **news**共现**10**次

单词 **learning** 和单词 **performance**共现**10**次

单词 **data** 和单词 **mining**共现**9**次

单词 **based** 和单词 **big**共现**8**次

单词 **based** 和单词 **data**共现**8**次

单词 **big** 和单词 **mining**共现**8**次

单词 **data** 和单词 **social**共现**8**次

单词 **learners** 和单词 **learning**共现**8**次

单词 **learners** 和单词 **performance**共现**8**次

单词 **based** 和单词 **information**共现**7**次

单词 **based** 和单词 **mining**共现**7**次

单词 **different** 和单词 **information**共现**7**次

单词 **information** 和单词 **large**共现**7**次

单词 **information** 和单词 **learning**共现**7**次

单词 **information** 和单词 **system**共现**7**次

单词 **IA** 和单词 **information**共现**6**次

实验评分：

指导教师签字：

年 月 日