**南 京 师 范 大 学**

**《数据挖掘课程》**

**实验报告**



**题 目： Apriori算法实现**

**专 业： 计算机科学与技术**

**姓 名： 蒋 泽**

**学 号： 19190248**

**计算机与电子信息学院/人工智能学院**

**2022**  **年 4 月 19 日**

目录

[目录 0](#_Toc101284425)

[一、 实验内容 2](#_Toc101284426)

[1. 要求 2](#_Toc101284427)

[2. 输入格式 2](#_Toc101284428)

[3. 实验环境 2](#_Toc101284429)

[二、 算法步骤 3](#_Toc101284430)

[1. 事物数据库的读取 3](#_Toc101284431)

[目标任务 3](#_Toc101284432)

[核心代码 4](#_Toc101284433)

[2. 频繁集的生成 5](#_Toc101284434)

[算法流程 5](#_Toc101284435)

[核心代码 6](#_Toc101284436)

[3．规则生成 8](#_Toc101284437)

[算法流程 8](#_Toc101284438)

[核心源码 8](#_Toc101284439)

[三、 实验结果 9](#_Toc101284440)

[测试1 9](#_Toc101284441)

[测试数据 9](#_Toc101284442)

[实验结果 10](#_Toc101284443)

[测试2 11](#_Toc101284444)

[测试数据 11](#_Toc101284445)

[实验结果 12](#_Toc101284446)

[测试3 13](#_Toc101284447)

[测试数据 13](#_Toc101284448)

[实验结果 14](#_Toc101284449)

[四、 实验总结 14](#_Toc101284450)

# 实验内容

## 1. 要求

1. 使用Java语言实现

2. 关联规则挖掘程序

## 2. 输入格式

事务数据库文件名称；  
示例文件格式：  
第一行：最小支持度 最小置信度  
第二行开始，每一行为一条购物记录，购物项目用空格分开  
“--END--”为结束符号

数据文件名称为：test.txt文件内容如下：  
0.2 0.3  
A B C  
D E A F  
C F B  
--END—

## 3. 实验环境

IDE环境：IDEA2020

JDK：JDK13

# 算法步骤

Apriori算法大概可以分为三步：（1）事物数据库的读取；（2）频繁集的生成；（3）强关联规则的生成。

|  |  |
| --- | --- |
| Apriori Algorithm | |
| 1: | Read transaction file |
| 2: | Frequent ItemSet generation |
| 3: | Association rule generation |

对应步骤：

public void apriori() {

*// 1，读取事务数据库，获取各种阈值*

read\_file("./src/test.txt");

*// 2.生成频繁集*

frequentSet\_gen();

printFreItemSets();

rule\_gen();

printf\_rule();

}

## 事物数据库的读取

### 目标任务

从事物数据文件中读取对应最小支持度min\_support,最小置信度min\_confidence，依次读取每一行作为一个transaction，遇到“—END--”结束。

### 核心代码

*/\*\**

*\* 读取文件*

*\**

*\** ***@param*** *input\_test\_file*

*\*/*

public void read\_file(String input\_test\_file) {

try {

BufferedReader br = new BufferedReader(new FileReader(new File(input\_test\_file)));

String line = "";

line = br.readLine();

min\_support = Double.valueOf(line.substring(0, line.indexOf(" ")));

min\_confidence = Double.valueOf(line.substring(line.indexOf(" ") + 1, line.length()));

while (!(line = br.readLine()).equals("--END--")) {

DB.add(line.split(" "));

}

br.close();

System.out.println("Input para==================================");

System.out.println("min\_support " + min\_support);

System.out.println("min\_confidence " + min\_confidence);

transcation\_num = DB.size();

System.out.println("transcation\_num " + transcation\_num);

System.out.println("transcation:==============================");

for (String[] T : DB) {

for (String item : T)

System.out.print(item + " ");

System.out.println();

}

System.out.println("========================================");

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

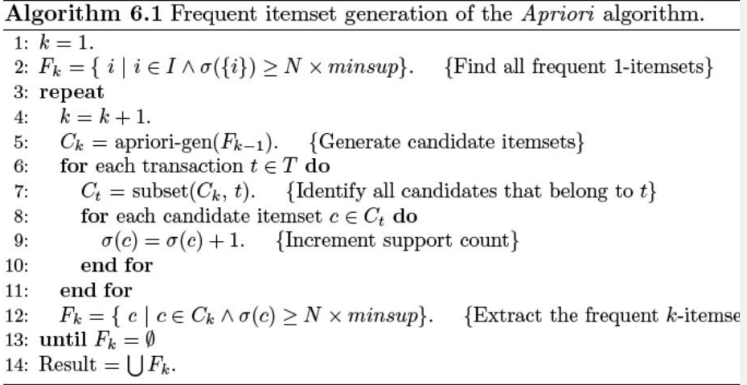
}

## 频繁集的生成

### 算法流程

#### 概览

如图，频繁图生成可以分为以下几步：



#### Apriori-gen

其中，第（5）步apriori-gen的生成方法有3种：  
 （1）由与生成  
 （2）由与生成  
 （3）由与生成

在apriori算法中选择实现算法（3）

原理为：

算法过程为：

1. 对于每一对，按照字典序排序与
2. 判断：若前k-2项均相等，则将其中一个的第k-1项移入另一个作为第k项，从而得到Ck

### 核心代码

（其余代码见源文件）

#### 频繁集生成核心代码

*/\*\**

*\* 频繁集的生成*

*\*/*

public void frequentSet\_gen() {

*// 首先生成L1*

FkList.add(get\_F1());

int k = 1;

while (FkList.get(k - 1).size() != 0) {

Map<String, Integer> Ck = get\_Ck\_from\_preFk(k);

*// 获取DB中的k项子集*

*// 计数Ck*

for (String[] transcation : DB) {

*// find ck in transcation*

Set<String> Ct = subset(Ck, transcation);

*// count Ck*

for (String item : Ct) {

Ck.put(item, Ck.get(item) + 1);

}

}

FkList.add(filter\_Ck2Fk(Ck));

k++;

}

}

#### 获得F1

*/\*\**

*\* 获得F1*

*\**

*\** ***@return***

*\*/*

private Map<String, Integer> get\_F1() {

Map<String, Integer> C1 = new HashMap<String, Integer>();

for (String[] Transcation : DB) {

if (Transcation.length > 1) {

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

if (C1.containsKey(Transcation[i])) {

C1.put(Transcation[i],

C1.get(Transcation[i]) + 1);

} else {

C1.put(Transcation[i], 1);

}

}

}

}

Map<String, Integer> F1 = filter\_Ck2Fk(C1);

return F1;

}

#### 由与生成的Apriori-gen算法

private Map<String, Integer> get\_Ck\_from\_preFk(int k) {

List<String> pre\_Fk\_list = new ArrayList<String>(FkList.get(k - 1).keySet());

Map<String, Integer> Ck = new HashMap<String, Integer>();

for (int i = 0; i < pre\_Fk\_list.size(); i++) {

for (int j = 0; j < pre\_Fk\_list.size(); j++) {

List<String> l1List = sortStrArray(pre\_Fk\_list.get(i));

List<String> l2List = sortStrArray(pre\_Fk\_list.get(j));

List<String> c = null;

if (isAttachable(l1List, l2List)) {

c = new ArrayList<String>(l1List);

c.add(l2List.get(l2List.size() - 1));

*// 判断生成的项集的所有子集是否都在Lk-1中，进行剪枝操作*

if (!isHasInfrequentSubset(c, k)) {

String strKey = strList2Str(c);

Ck.put(strKey, 0);

}

}

}

}

return Ck;

}

## 3．规则生成

### 算法流程

规则生成方法此处采用的是暴力直接生成法：

1. 对于每个项数大于1的频繁集，求出其所有非空真子集
2. 对于每个非空真子集，同每一条transaction进行比较，若为transaction的子集，则count+1
3. 计算对应非空真子集的support与频繁集的support，从而得出confidence，判断是否为强关联规则

### 核心源码

private void rule\_gen() {

*// 对于每一个大于二的频繁集，首先生成频繁集的所有子集*

for (Map<String, Integer> frequentItemSets : FkList) {

Iterator<Map.Entry<String, Integer>> iter = frequentItemSets.entrySet().iterator();

while (iter.hasNext()) {

Map.Entry<String, Integer> entry = iter.next();

if (entry.getKey().length() <= 1)

*// 对于频繁集F1不做处理*

continue;

String frequentItem = entry.getKey();

*// 生成frequentItem的所有子集，而后对于每一个子集，生成规则的置信度*

ArrayList<ArrayList<String>> subSets = get\_all\_subSet(frequentItem.split(" "));

for (ArrayList<String> subSet : subSets) {

if (isRule(subSet, frequentItem)) {

rule.put(subSet, get\_remain\_subSet(subSet, frequentItem.split(" ")));

}

}

}

}

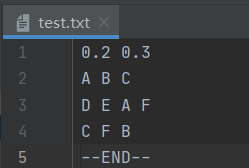
}

# 实验结果

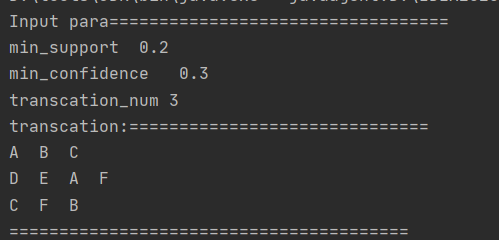
运行说明，老师要运行代码时，此处考虑测试文件与源文件同一目录下，若不同环境下更改位置，请修改对应输入路径

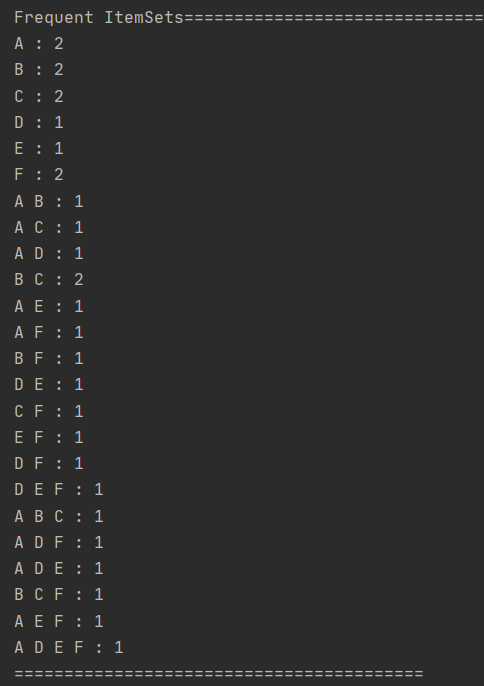
## 测试1

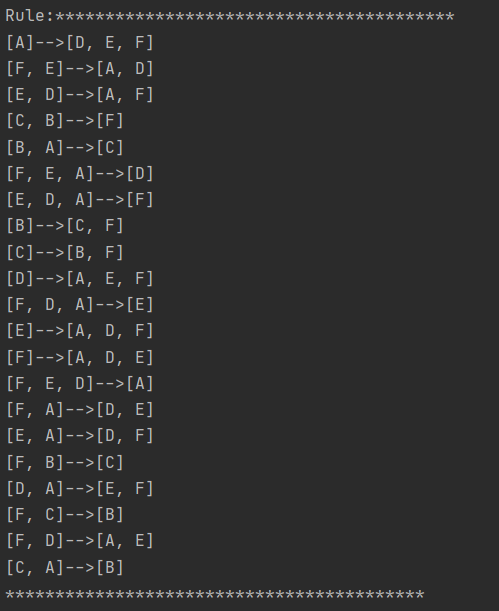
### 测试数据



### 实验结果

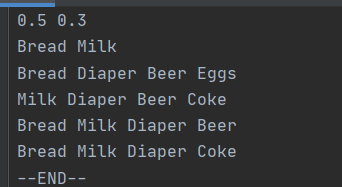




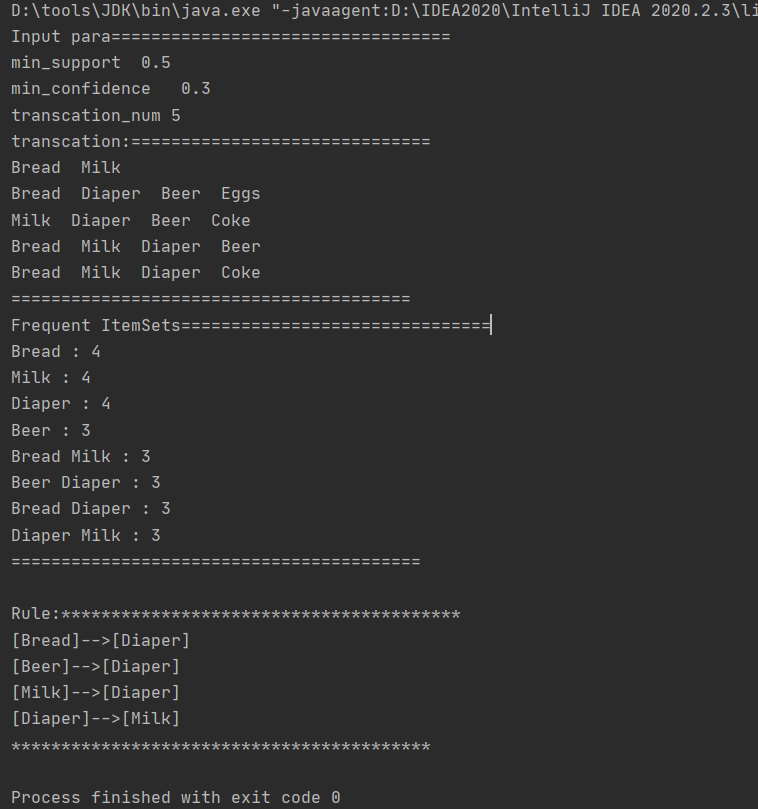


## 测试2

### 测试数据

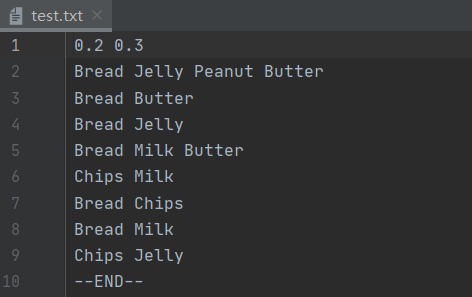


### 实验结果

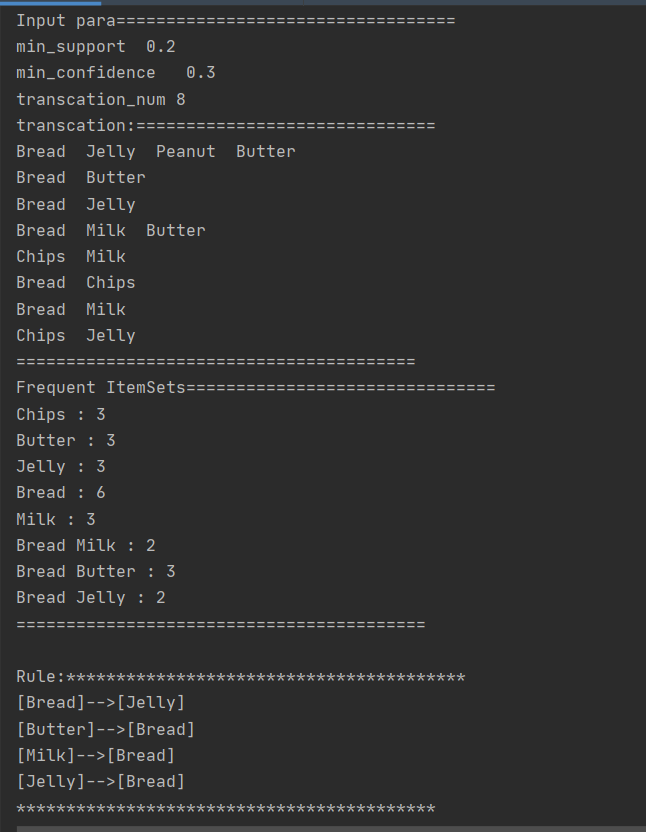


## 测试3

### 测试数据



### 实验结果



# 实验总结

挖掘关联规则的核心是频繁集的获取与关联规则的获取，而这其中的要达到的目标是“好”与“快”，尤其是“快”，Apriori算法作为数据挖掘的十大算法之一，利用先验原理来根据生成的方式与基于与相比降低了计算量，通过剪枝方式减少了非必要的候选集的计算；在规则生成中，可以利用hashTree来减少规则生成的计算量，即使是暴力的规则生成方式，也可以利用单调性的特点来剪枝从而减少计算量。

实验结果中，虽然根据规则生成了许多符合要求的频繁集与规则，但是算法是死的，这些规则在生活中可能是没有意义的，比如，若B的支持度是50%，而通过计算，发现规则A->B的置信度是40%，假设高于最低置信度，是一条强关联规则，但是这样的规则是没有现实意义的，（没有必要给购买A的人推荐B,他本身购买B的概率就很大的，没必要做无谓的推荐）所以，能够单独依靠一个算法达到人的思考的层次、效果是十分困难的，只有多思考，多优化，多维度的思考才能更加体现算法的意义。