**电子科技大学计算机科学与工程学院**

**标 准 实 验 报 告**

**（实验）课程名称 数据结构与算法**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：陶浩轩 学 号：2023080902011 指导教师：陈端兵**

1. **实验室名称：**

学知三组团10栋139

**二、实验项目名称：**

频繁模式挖掘

**三、实验原理：**

实验基于 Apriori 算法，利用先验性质，通过连接和剪枝操作，从给定的数据集中挖掘出频繁项集和关联规则。

**四、实验目的：**

1. 理解频繁模式挖掘的概念和重要性。

2. 掌握 Apriori 算法的基本原理和实现方法。

3. 利用 Apriori 算法挖掘数据集中的频繁项集和关联规则。

4. 分析频繁模式挖掘的结果，并进行解释和应用。

**五、实验内容：**

数据准备： 随机生成指定范围的数据

参数设置： 设置支持度阈值，确定挖掘频繁项集的最小支持度。

算法实现： 编写 Apriori 算法的代码，实现频繁项集的挖掘。

结果分析： 分析挖掘出的频繁项集和关联规则，并进行解释和应用。

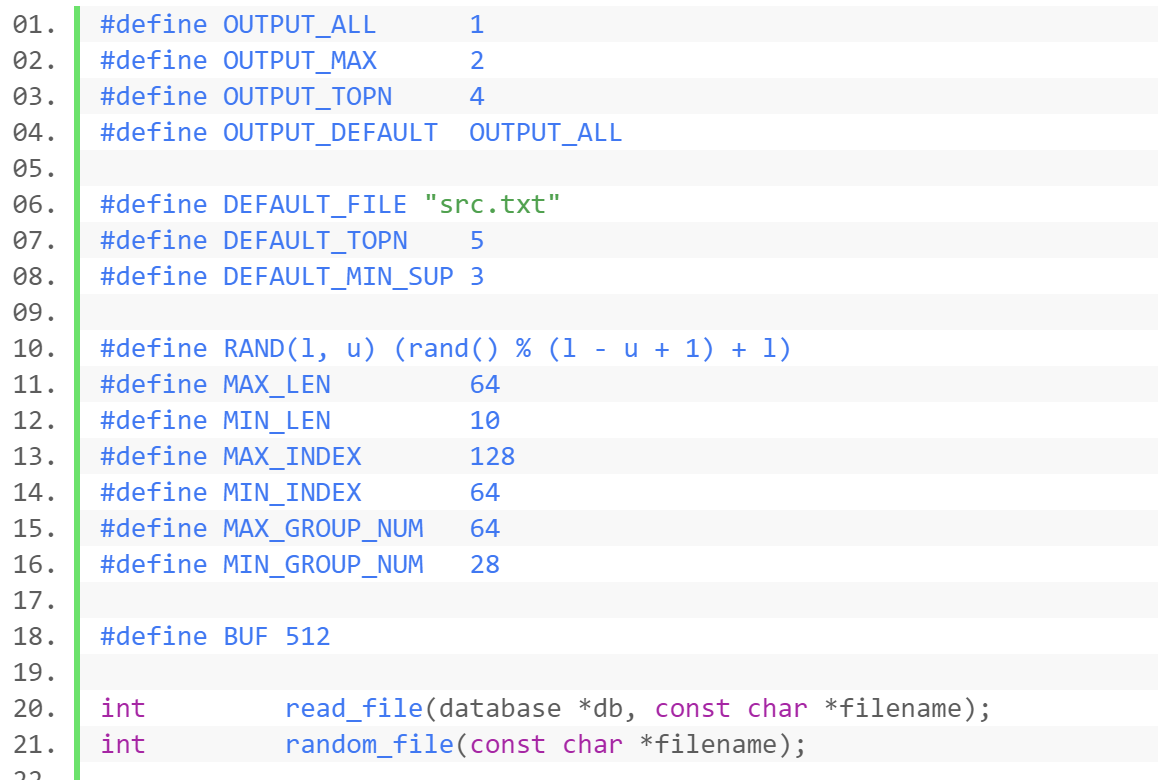
**六、实验器材（设备、元器件）：**

平台：WSL Ubuntu 22.04 on Windows 10

**七、实验步骤：**

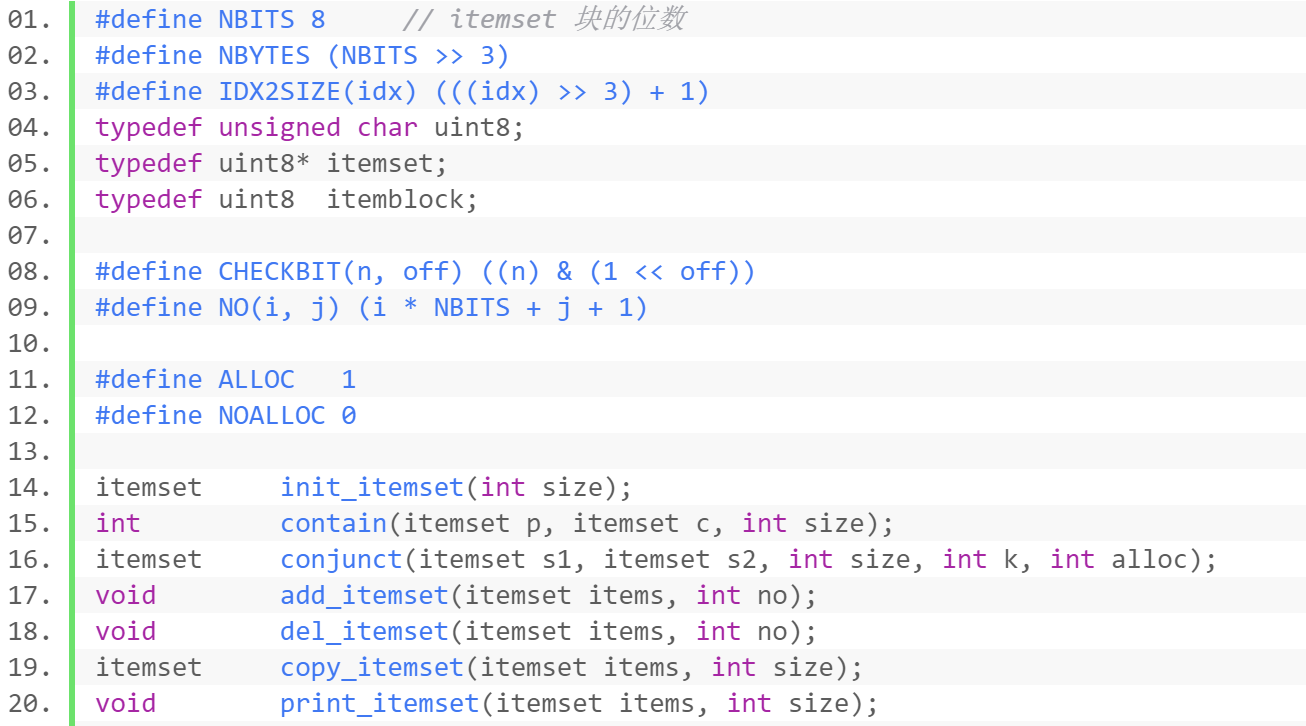
代码实现在 https://github.com/WhaleFall-UESTC/UESTC-DataStructure/tree/master/proj1

1. 定义 file，随机生成数据以及将文件读取为 database，并设置基本参数（随机变量以及最小支持度的范围）



file.h

2. 定义项集的存储方式：类似于 bitmap，定义 itemset结构，描述从某一地址开始一定长度的比特串，其中从开始的第 i 位的比特，1 表示编号 i 的元素存在于该项集，0 表示不存在，并定义 freqitem 与freqlist 结构，以链表的形式存储 itemset

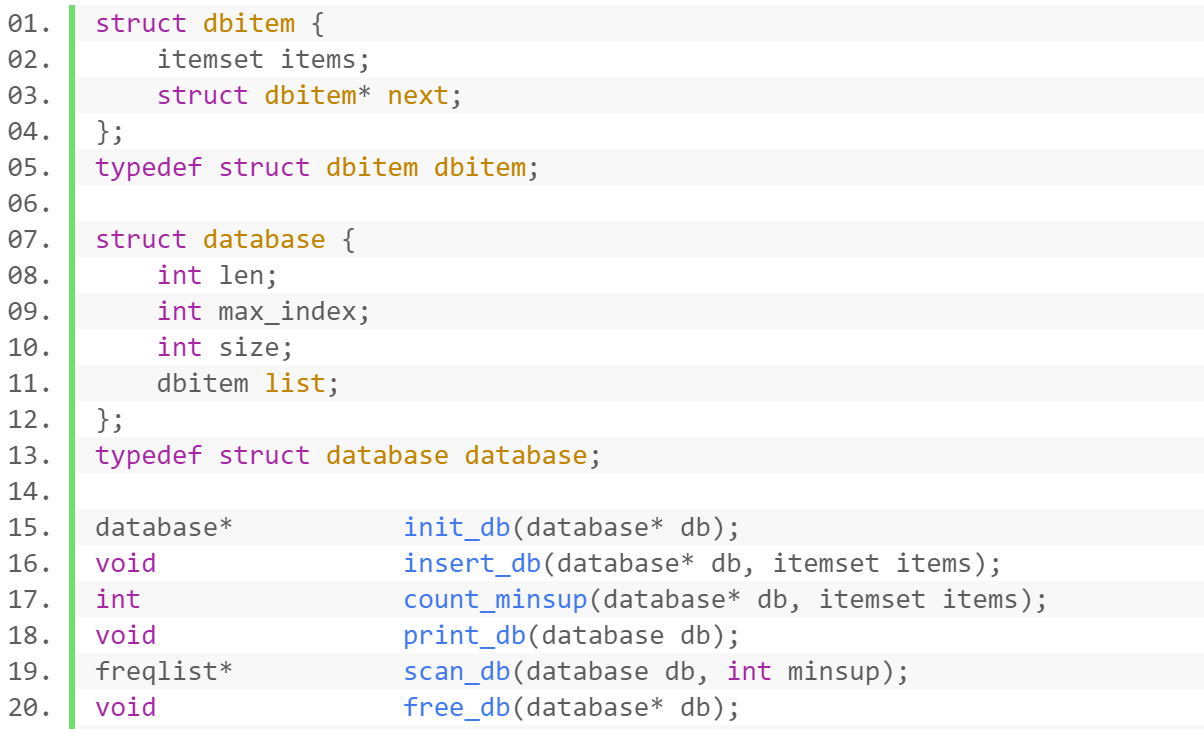


itemset.h

****

freqlist.h

3. 定义 database 结构，此结构存储最初始的项集

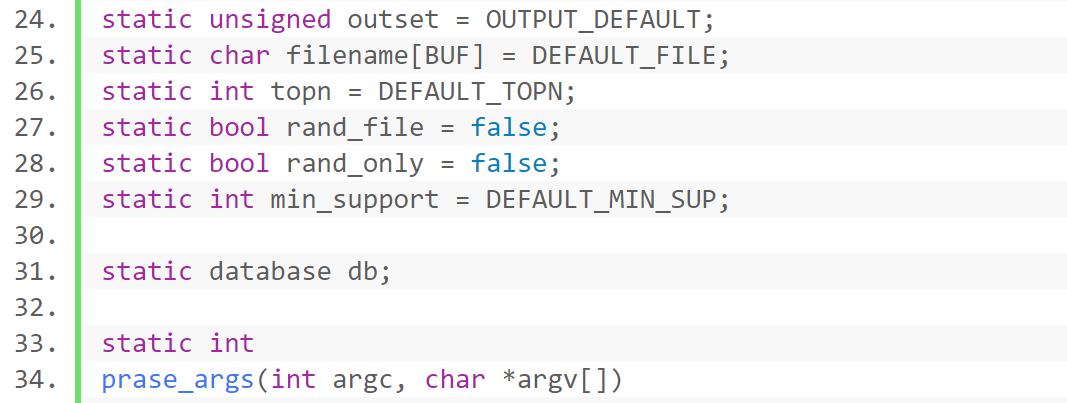


database.h

4. 编写 debug.h 与 Makefile 方便调试与测试

5. 基础设施准备完毕，单元测试各模块后，开始在 main 函数实现 Apriori 算法

定义全局变量 db 以及与结果输出，文件读取相关的参数：



Apriori 算法的连接与剪枝操作。虽然从概念上这两个操作是解耦的，但是一些过滤需要二者的过程信息，出于效率考虑，我选择将它们合并在一起



在 main 函数中，先解析参数，生成随机数据，读取数据并建立 database，并以此获取第一个 k = 1 的频繁项集，随后不断调用 link\_with\_cut 筛选



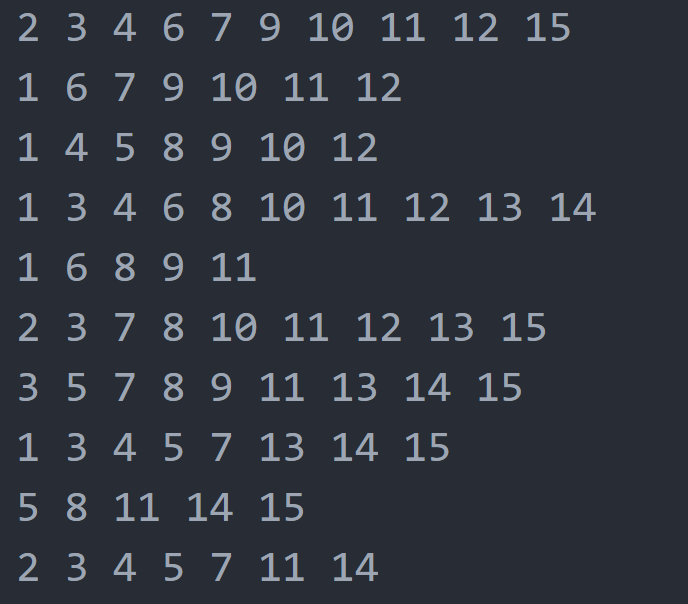
后续再处理输出结果。为了按照从小到大的顺序输出给定规则的频繁项集，我构建另一个变长的最小堆来存储 Apriori 每一轮的结果



至此，初步实现了 Apriori 算法

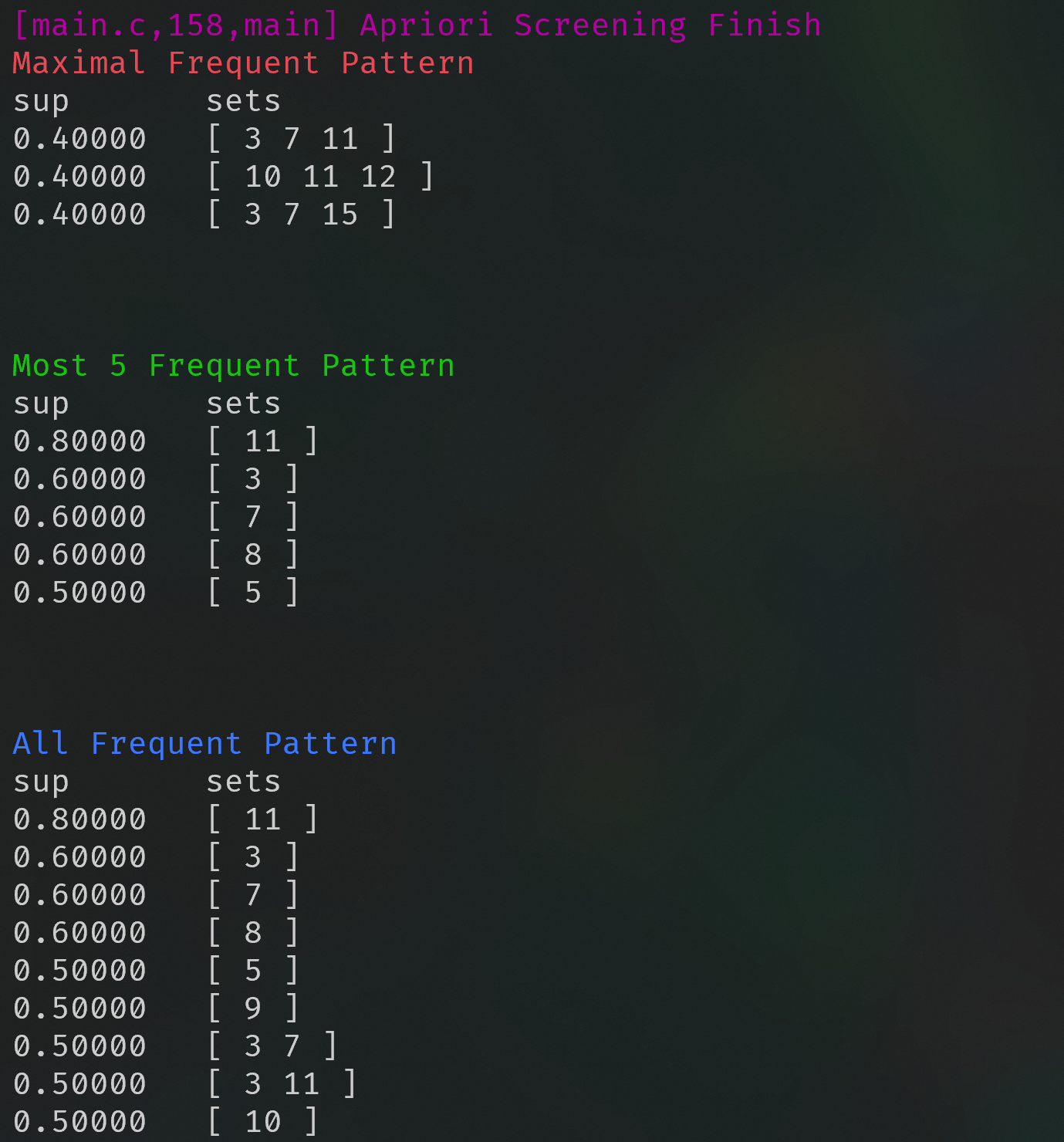
**八、实验数据及结果分析：**

我生成了一个小的数据集来进行测试，设置 min\_support = 0.4



这些数据可以表示一个图书馆内的部分顾客借阅过的书籍的类型

通过运行程序，我们可以得到一下的结果：



我们可以通过 Apriori 算法的到频繁项集，可以得知有哪些编号的关联性大

比如 3 类型代表犯罪心理学书籍，7 代表侦探小说，我们可以发现这两类书被同一人借阅的概率大，于是我们可以将这两类书的书架搬近一点

**九、实验结论：**

Apriori 算法是数据挖掘方面一个经典的算法。通过合并，剪枝，不断地筛选来找到频繁项集。我们可以通过这个方法来获取大量数据中的频繁模式，但是其平方时间复杂度使得其效率有些堪忧（一些较大但有没那么大的数据集甚至跑到了十几秒）。在使用的过程中，我们应当多采取一下优化的措施，改良算法

**十、总结及心得体会：**

1. 本次我使用了 bitmap 的形式存储频繁项集，不仅节省了空间（虽然Apriori更关心时间复杂度），而且在判断是否为子集时只需要将二者 and，合并时只需要将二者 or，操作上也更加快捷

2. 我尝试设计了一套层级结构：freqitem 与 freqlist 建立在 itemset 之上；database 建立在 freqlist 之上，这套层级结构方便了程序的构建

**十一、对本实验过程及方法、手段的改进建议：**

1. 我们需要在 database 中查找频繁项集出现的次数，采用的是顺序查找的方式。优化的方案是：我们可以利用哈希表来减小查找次数。根据 itemset 的长度，我们可以按照偏移量均匀地选取 4 个比特，并用这 4 个比特所组成的无符号数的值作为 hashcode
2. 如果 itemset 的长度较小（小于 128），那么这个 itemset 是对 C 而言一个天然的无符号数表示。我们可以在哈希桶内采用二分查找（或者构建二叉搜索树）的方式来减小搜索次数。此外，还可以根据 itemset 的长度与数量的大小决定使用基数排序还是快速排序
3. 对应的，在 database 中可以采用变长数组的方式而非链表（即初始长度固定，若所要的容量超过了这个长度，则再开辟一个大小为原来的两倍的空间，将数据复制到这个空间，释放原来的空间），可以加快查找速度
4. 或者说构建一棵树，以一个项集的第一个元素作为根，若是其他项集有与其共同的部分，但是后续的项目有所不同，则在这棵树上“分支”，并且树要求子节点必须按照某一顺序大于根节点。这样在计数的时候可以快速地获取支持度
5. 最开始写代码的时候可以参照网上的测试集的格式来编写
6. 再分析优化程序时，可以使用 ftrace 跟踪函数调用与开销，针对性地优化

**报告评分：**

**指导教师签字：**