解题代码

```
# 根据Apriori算法步骤，编写代码实现寻找频繁项集
#  (1) 首先，定义create_C1函数，通过扫描数据集创建大小为1的所有候选项集的集合


def create_C1(data_set):
    C1 = set()
    for t in data_set:
        #*************** BEIGN ******************
        for item in t:
                    item_set = frozenset([item])  # frozenset: 创建不可变集合
                    C1.add(item_set)
        #**************** END ******************
    return C1

#  (2) 定义is_apriori函数来判断候选K项集是否满足先验原理.其中，CK_item是CK中的一个候选k项集；Lksub1是包含全部的频繁(k-1)项集的集合。
# 如果不满足先验原理（即一个候选k项集Ck-item的(k-1)项子集不在Lksub1中），则返回False，如果满足则返回True。


def is_apriori(Ck_item, Lksub1):
    for item in Ck_item:
        #*************** BEIGN ******************
        sub_Ck = Ck_item - frozenset([item])
        if sub_Ck not in Lksub1:
            return False
        #**************** END ******************

    return True

# (3) 定义create_CK函数，由频繁(k-1)项集的集合Lksub1的自身连接产生候选k项集Ck。
#其中，对于不满足先验原理的Ck_item，进行剪枝，得到最终的候选K项集。其中，Lksub1包含频繁(k-1)项集的集合，k是频繁项集的项数
# apriori_gen(Fk-1)apriori_gen(Fk-1)


def create_Ck(Lksub1, k):  # ((A,B),(A,D),(C,D))
    Ck = set()
    len_Lksub1 = len(Lksub1)
    list_Lksub1 = list(Lksub1)  # [(A,B),(A,D),(C,D)]
    for i in range(len_Lksub1):
        for j in range(1, len_Lksub1):
            l1 = list(list_Lksub1[i])  # [A,B]
            l2 = list(list_Lksub1[j])  # [A,D]
            l1.sort()  # 排序
            l2.sort()  # 排序
            #**************** BEGIN ******************
            if l1[0:k-2] == l2[0:k-2]:  # 如果list_Lksub1中的前一项等于它的后一项
                Ck_item = list_Lksub1[i] | list_Lksub1[j]  # (A,B) | (A,C)
                if is_apriori(Ck_item, Lksub1):  # 对于满足先验原理，加入频繁项集
                    Ck.add(Ck_item)
            #***************** END ******************
    return Ck  # ((A,B,C),(...))


# (4) 定义generate_Lk_by_Ck函数，通过从Ck执行删除策略来生成Lk，即对Ck中的每个项进行计数，然后删除不满足最小支持度的项，从而获得频繁k项集。Lk：包含所有频繁k项集的集合

def generate_Lk_by_Ck(data_set, Ck, min_support, support_data):
```

```
    Lk = set()
    item_count = {}
    ################ BEGIN ###################
    for t in data_set:
        for item in Ck:
            if item.issubset(t):  # 判断集合的所有元素是否都包含在指定集合中，如果是则返回 True，否则返回 False。
                # 更新项集的支持度计数
                if item not in item_count:
                    item_count[item] = 1
                else:
                    item_count[item] += 1
                ################  END  ###################
    t_num = float(len(data_set))

    for item in item_count:
        if (item_count[item] / t_num) >= min_support:
            Lk.add(item)
            ################ BEGIN ###################
            #存储每个频繁项集的支持度
            support_data[item] = item_count[item] / t_num
            ################  END  ###################
    return Lk

#  (5) 定义generate_L获取频繁项集。参数k是所有频繁项集的最大项数，min_support是最小支持度，data_set是要挖掘的数据集（整合前面几个函数，得到频繁项集）


def generate_L(data_set, k, min_support):
    support_data = {}
    C1 = create_C1(data_set)  # 创建候选1-项集的集合
    L1 = generate_Lk_by_Ck(data_set, C1, min_support,
                           support_data)  # 从候选1-项集得到频繁1-项集
    Lksub1 = L1.copy()  # 浅复制
    L = []
    L.append(Lksub1)
    ################ BEGIN ###################
    for i in range(2, k+1):
        Ci = create_Ck(Lksub1, i)  # apriori_gen(Fk-1)
        Li = generate_Lk_by_Ck(data_set, Ci, min_support, support_data)  # 从候选i-项集得到频繁i-项集
        Lksub1 = Li.copy()
        L.append(Lksub1)
    ################ END ###################
    return L, support_data  # L所有频繁项集，support_data所有频繁项集的支持度


# 调用编写的函数，寻找本实验数据集中的频繁项集

def load_data(filename):
    data = list()
    with open(filename, 'r', encoding='utf-8') as f:
        for line in f.readlines():
            linestr = line.strip()
            linestrlist = linestr.split(",")
            data.append(linestrlist)
    return data


import fpGrowth as fp
```

```python
def Task():
    data = load_data('data/apriori.txt')
    #**************** BEGIN *****************
    L, support_data = generate_L(data, k=3, min_support=0.06)
    #**************** END ******************

    sum = 0
    for Lk in L:  # Lk是k-频繁项集
        sum += len(Lk)
    len_number = sum  # 输出频繁项集的总数

    # 验证结果可靠性，将FP算法的结果与Apriori算法结果进行对比
    # 从之前的实验结果可知，FP算法和Apriori算法寻找的频繁项集的数量是相同的，均为209个，频繁项集的最大项数都是3。
    # 比较FP算法和Apriori算法得到的频繁项集是否相同

    result = fp.getFrequentItemsets()

    q = set(frozenset(itemset) for itemset, supports in result)  # FP树的频繁项集集合
    p = set(freqlist for Lk in L for freqlist in Lk)  # Apriori算法的频繁项集集合

    return len_number, q,p
```