Homework 5.2 Report

本实验实现了FP_growth算法，并使用交易数据集验证算法结果的正确性

FP_growth算法实现代码

算法使用python实现，本节对代码框架进行解释

主函数：

```
if __name__ == '__main__':
    MIN_SUPPORT = 2
    transactions = []

    BASE_DIR = os.path.dirname(__file__)
    file_path = os.path.join(BASE_DIR, "homework.dat")
    data = []
    dataIn = open(file_path, 'r')

    for line in dataIn:
        line = line.strip();
        index, content = line.split(' ')
        content = content.split(',')
        transactions.append(content)

    result = []
    for itemset, support in find_frequent_itemsets(transactions, MIN_SUPPORT, True):
        result.append((itemset,support))

    result = sorted(result, key=lambda i: i[0])
    for itemset, support in result:
        print str(itemset) + ' ' + str(support)
```

MIN_SUPPORT为最小支持度，transactions为输入数据集（每行数据在其中以列表形式存在），dataIn为输入文件流，result为结果集（包含项集和频数）

其它函数及类：

- def find_frequent_itemsets(transactions, minimum_support, include_support=False):

  transactions表示数据集，minimum_support表示最小支持度，include_support限制是否记录项集频数，该函数返回所求频繁项集

- class FPTree(object):

  FP树的结构类，包含如下函数

  - def root(self):

    返回根节点

  - def add(self, transaction):

    添加节点

  - def _update_route(self, point):

    更新point结点的路径

  - def items(self):

返回当前节点和他的孩子节点

- def nodes(self, item):

  返回包含item的结点

- def prefix_paths(self, item):

  生成以item为末尾的后缀路径

- def inspect(self):

  打印树的结构信息

- def conditional_tree_from_paths(paths): 通过给定路径paths生成条件FP树
- class FPNode(object): FP结点，包含以下方法：
  - def **init**(self, tree, item, count=1):

    初始化节点信息

  - def add(self, child):

    将child添加为self的子节点

  - def search(self, item):

    查找self是否包含子节点item

  - def **contains**(self, item):

    查找self是否包含子节点item

作业运行结果

- ['1'] 6
- ['1', '3'] 4
- ['1', '5'] 2
- ['2'] 7
- ['2', '1'] 4
- ['2', '1', '3'] 2
- ['2', '1', '5'] 2
- ['2', '3'] 4
- ['2', '4'] 2
- ['2', '5'] 2
- ['3'] 6
- ['4'] 2
- ['5'] 2

FP-growth vs. Apriori

相比于Apriori多次搜索输入集合并枚举得到频繁项集，FP-growth利用树形结构直接得到频繁项集，减少扫描输入的次数节省时间。随着支持度的降低，FP-growth运行时间增长的幅度明显小于Apriori算法。但是由于树形结构、多次构件条件树等开销，FP-growth算法的空间开销相对较大。