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

一、 实验室名称：

学知三组团 110139

二、实验项目名称：

频繁模式与关联规则挖掘

二、 实验原理：

FP 树通过将数据集中的频繁项集压缩成树形结构，仅存储满足最小支持度的项，从而能够高效地挖掘频繁项，因为它减少了冗余数据，并允许通过树结构快速计算项集的支持度。

四、实验目的：

通过构建频繁模式树（FP-tree），挖掘数据集中的频繁项集和关联规则，以了解项目之间的关联性。

五、实验内容：

1. 构建 FP-tree
2. 使用 FP-Growth 挖掘频繁项集

六、实验器材（设备、元器件）：

Windows 10，Python 3.12.6

七、实验步骤：

1. **读取数据集：**

   调用 readcsv 函数，传入事务数据库的文件路径，打开 CSV 文件，逐行读取内容，将每个项目转换为整数类型，并将每个事务表示为一个元组（包含支持度计数和项目集列表），最后将所有事务存储在一个列表中并返回。

2. **初始化 FP 树：**

   创建 FPTree 类的一个实例，初始化根节点（其 nid 设为-1，支持度为 0），同时初始化一个空的事务数据库和头表，为构建 FP 树做准备。

3. **构建 FP 树：**

   调用 build 方法，传入事务数据库文件路径或已经读取的事务列表，遍历每个事务，按支持度降序处理项目集，更新 FP 树和头表，对于每个项目，如果在当前节点的子节点中已存在，则增加其支持度，否则创建新节点，并将其添加到子节点和头表中。

4. **递归增长 FP 树以挖掘频繁项集：**

   调用 growth 方法，传入最小支持度阈值，若最小支持度为小数，则转换为事务总数的整数比例，遍历头表中的每个项目，若其支持度大于等于最小支持度，则递归构建条件模式树，并挖掘出频繁项集。

5. **递归挖掘频繁模式：**

   fp_growth 方法用于递归挖掘频繁模式，若树为单路径，则直接提取路径中的所有组合作为频繁项集；否则，遍历头表中的每个项目，构建条件模式，并递归挖掘。

```
Procedure FP-growth (Tree, α)
{
(1) if Tree contains a single path P
(2) then for each combination (denoted as β)
    of the nodes in the path P do
(3)     generate pattern β ∪ α with support =
            minimum support of nodes in β;
(4) else for each a_i in the header of Tree do {
(5)     generate pattern β = a_i ∪ α with
            support = a_i.support;
(6)     construct β's conditional pattern base and
            then β's conditional FP-tree Tree_β;
(7)     if Tree_β ≠ ∅
(8)     then call FP-growth (Tree_β, β)          }
}
```

6. **获取条件模式：**

   cond_pattern 方法用于获取给定节点的条件模式，通过从当前节点向上遍历至根节点，记录路径上的节点 ID，形成条件模式。

7. **打印 FP 树和头表：**

   print 方法用于打印 FP 树的结构，展示树中每个节点的 ID 和支持度；print_header_table 方法用于打印头表，显示每个项目的支持度和链接的节点。

8. **挖掘并打印最大频繁项集：**

   调用 mine 函数，传入事务数据库文件路径和最小支持度，构建 FP 树，挖掘频繁项集，并按支持度降序排序结果；print_maximal 函数用于打印前 n 个最大频繁项集。

9. **主程序执行：**

   在 if __name__ == '__main__': 块中，解析命令行参数以获取文件路径和最小支持度，调用 mine 函数执行频繁模式挖掘，并打印挖掘出的频繁项集。

八、实验数据及结果分析：

使用数据集：transactional_T10I4D100K.csv，共有 100000 笔交易

```
F:\Projects\Data Structrue & Algorithm\UESTC-DataStructure\proj4
❯ & D:/Python312/python.exe "f:/Projects/Data Structrue & Algorithm/UESTC-DataStructure/pr
oj4/FP.py" .\test\transactional_T10I4D100K.csv 0.02
2000
[(7828, [368]), (7057, [529]), (6810, [829]), (6265, [766]), (5845, [722]), (5835, [354]),
 (5408, [684]), (5375, [217]), (5102, [494]), (5057, [419]), (4993, [692]), (4973, [120]),
 (4902, [883]), (4681, [937]), (4629, [177]), (4559, [145]), (4511, [438]), (4438, [460]),
 (4388, [362]), (4309, [789]), (4258, [39]), (4248, [32]), (4137, [470]), (4132, [682]), (
4082, [283]), (4037, [914]), (3982, [538]), (3921, [966]), (3883, [183]), (3864, [720]), (
3771, [775]), (3735, [541]), (3710, [919]), (3690, [947]), (3686, [888]), (3667, [401]), (
3649, [862]), (3626, [956]), (3605, [205]), (3507, [71]), (3470, [346]), (3420, [489]), (3
415, [12]), (3385, [895]), (3361, [795]), (3281, [510]), (3219, [598]), (3151, [75]), (313
5, [487]), (3134, [614]), (3103, [798]), (3090, [8]), (3085, [825]), (3069, [350]), (3063,
 [793]), (3053, [886]), (3044, [509]), (3043, [885]), (3014, [279]), (3012, [918]), (2976,
 [675]), (2959, [381]), (2943, [581]), (2902, [571]), (2894, [471]), (2883, [597]), (2860,
 [758]), (2852, [72]), (2847, [854]), (2847, [694]), (2843, [548]), (2835, [569]), (2814,
[844]), (2810, [871]), (2794, [944]), (2793, [631]), (2793, [526]), (2791, [175]), (2783,
[561]), (2777, [93]), (2767, [782]), (2743, [57]), (2742, [239]), (2732, [960]), (2725, [5
22]), (2717, [676]), (2713, [998]), (2693, [661]), (2687, [140]), (2685, [390]), (2684, [7
97]), (2680, [112]), (2668, [606]), (2666, [21]), (2641, [132]), (2637, [413]), (2634, [65
3]), (2628, [274]), (2618, [236]), (2614, [617]), (2611, [151]), (2601, [593]), (2600, [28
5]), (2595, [54]), (2578, [752]), (2527, [674]), (2514, [778]), (2492, [634]), (2479, [276
]), (2472, [48]), (2471, [78]), (2462, [477]), (2425, [921]), (2420, [392]), (2411, [70]),
 (2402, [38]), (2386, [788]), (2370, [69]), (2354, [663]), (2325, [242]), (2320, [161]), (
2309, [480]), (2306, [780]), (2288, [638]), (2281, [229]), (2244, [523]), (2237, [874]), (
2237, [803]), (2210, [296]), (2193, [116]), (2179, [73]), (2177, [744]), (2174, [204]), (2
165, [27]), (2164, [579]), (2163, [809]), (2149, [6]), (2146, [334]), (2129, [738]), (2125
, [472]), (2108, [280]), (2100, [620]), (2096, [196]), (2089, [387]), (2086, [970]), (2082
, [450]), (2062, [832]), (2047, [878]), (2047, [411]), (2046, [774]), (2041, [349]), (2022
, [826]), (2009, [210]), (2007, [373]), (2004, [192])]

[16:25] Shell master ≡ ?5 ~2 -3                                          7s 749ms
F:\Projects\Data Structrue & Algorithm\UESTC-DataStructure\proj4
❯
```

可以看到，相较于实验一提供的 Apriori 算法，FP-Tree 可以在更短的时间之内完成更加巨量的数据的挖掘。

九、实验结论：

通过本次实验，我们验证了 FP-Tree 算法在频繁模式与关联规则挖掘中的有效性和高效性。实验结果表明，FP-Tree 算法不仅能够快速挖掘出频繁项集，而且能够有效地生成关联规则，为数据分析和决策提供了有力的支持

十、总结及心得体会：

在本次实验中，我深入理解了 FP-Tree 算法的原理和实现过程，并通过实践操作掌握了如何使用该算法进行频繁模式挖掘。实验过程中，我体会到了数据结构算法在优化方面的重要性。高效的数据结构是提升处理效率的一大关键。

十一、对本实验过程及方法、手段的改进建议：

1. 可以进一步针对数据集进行数据结构方面的优化，比如针对数据的范围设定特定大小的数据类型
2. 在读取数据集和构建 FP 树的过程中，可以利用多线程或分布式计算来并行处理数据，提高构建速度
3.

报告评分：

指导教师签字：

附录：源代码

FP.py

```
import csv
import sys

def readcsv(filepath: str) -> list[tuple[int, list[int]]]:
    ret = []
    with open(filepath, newline='') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        for row in reader:
            digit = []
            for item in row:
                digit.append(int(item))
            ret.append((1, digit))
    return ret


class FPNode:
    def __init__(self, nid, support):
        self.nid = nid;
        self.support = support
        self.next = None
        self.parent = None
        self.children = {}
        self.visited = False

    def single_path(self):
        size = len(self.children)
        if (size == 0): return True
        elif (size == 1): return self.first_child().single_path()
        else: return False

    def print_tree(self, level=0):
        print(' ' * level * 4 + f'Node ID: {self.nid}, Support: {self.support}')
        for child in self.children.values():
            child.print_tree(level + 1)

    def first_child(self):
        children = list(self.children.values())
        if (children):
            return children[0]
        else:
            return None;

    def get_combinations(head):
        if not head:
            return [[]]
```

```
        rest_combinations = FPNode.get_combinations(head.first_child())
        current_combinations = [[head.nid] + combination for combination in
rest_combinations]
        return current_combinations + rest_combinations

    def find_min_support(self):
        min_sup = 1 << 31
        ptr = self.first_child()
        while (ptr):
            if ptr.support < min_sup:
                min_sup = ptr.support
            ptr = ptr.first_child()
        return min_sup


class FPTree:
    ROOT_NID = -1
    TABLE_NID = 0
    TABLE_SUPPORT = 1
    TABLE_HEAD = 2
    TABLE_TAIL = 3

    def __init__(self):
        self.root = FPNode(FPTree.ROOT_NID, 0)
        self.database = []
        self.header_table = []  # [nid, support, head, tail]

    def build(self, input):
        if isinstance(input, str):
            self.build_by_db(readcsv(input))
        else:
            self.build_by_db(input)

    def build_by_db(self, db: list[tuple[int, list[int]]]):
        self.database = db
        self.N = len(db)
        header_table = {}
        for support, itemset in db:
            current_node = self.root
            for nid in itemset:
                # build FP tree
                if nid in current_node.children.keys():
                    current_node.children[nid].support += support
                    header_table[nid][FPTree.TABLE_SUPPORT] += support
                else:
                    new_node = FPNode(nid, support)
                    new_node.parent = current_node
                    current_node.children[nid] = new_node
                    # build head table
                    if nid in header_table.keys():
                        header_table[nid][FPTree.TABLE_SUPPORT] += support
                        header_table[nid][FPTree.TABLE_TAIL].next = new_node
```

```
                    header_table[nid][FPTree.TABLE_TAIL] = new_node
                else:
                    header_table[nid] = [nid, support, new_node, new_node]

            current_node = current_node.children[nid]

        self.header_table = sorted(header_table.values(), key = lambda entry :
(entry[FPTree.TABLE_SUPPORT], -entry[FPTree.TABLE_NID]))

    def growth(self, min_sup):
        if isinstance(min_sup, float):
            min_sup = int(self.N * min_sup)
            print(min_sup)
        result = []
        FPTree.fp_growth(self, [], min_sup, result)
        return result

    def fp_growth(tree, a, min_sup, result):
        if tree.root.single_path():
            combinations = FPNode.get_combinations(tree.root.first_child())
            support = tree.root.find_min_support()
            for b in combinations:
                result.append((support, a + b))
        else:
            for ai in tree.header_table:
                # filter who is not frequent
                support = ai[FPTree.TABLE_SUPPORT]
                if support < min_sup:
                    continue
                b = a.copy()
                b.append(ai[FPTree.TABLE_NID])
                result.append((support, b))
                # go through link list
                ptr = ai[FPTree.TABLE_HEAD]
                prefix = []
                while (ptr):
                    # get node prefix
                    cpb = FPTree.cond_pattern(ptr)
                    if (cpb):
                        prefix.append(cpb)
                    ptr = ptr.next

                if (prefix):
                    treeb = FPTree()
                    treeb.build_by_db(prefix)
                    if (treeb.root.children):
                        FPTree.fp_growth(treeb, b.copy(), min_sup, result)


    def cond_pattern(ptr: FPNode):
        prefix = []
        support = ptr.support;
        ptr = ptr.parent;
```

```
        while (ptr.nid != FPTree.ROOT_NID):
            prefix.append(ptr.nid)
            ptr = ptr.parent
        return (support, prefix[::-1])


    def print(self):
        self.root.print_tree()
        print()
        self.print_header_table()

    def print_header_table(self):
        for item in self.header_table:
            print("(nid: {}, sup: {})".format(item[FPTree.TABLE_NID],
item[FPTree.TABLE_SUPPORT]), end=' ')
            ptr = item[FPTree.TABLE_HEAD]
            while (ptr) :
                print('-> (sup: {})'.format(ptr.support), end=' ')
                ptr = ptr.next;
            print()



def mine(csvfile, min_sup):
    tree = FPTree()
    tree.build(csvfile)
    result = tree.growth(min_sup)
    return sorted(result, key = lambda entry : -entry[0])


if __name__ == '__main__':
    assert len(sys.argv) >= 3
    file = sys.argv[1]
    min_support = float(sys.argv[2])
    print(mine(file, min_support))
```