关联规则挖掘——FP-growth 算法

2153726 罗宇翔

(经济与管理学院, 信息管理与信息系统)

摘　要

前文对关联规则挖掘的鼻祖算法 *Apriori* 进行了介绍，在复杂度分析中提到，*Apriori* 算法的时间复杂度为平方级，空间复杂度为指数级。本文将介绍另一种算法（$FP-growth$），它在时间和空间复杂度上都优于 *Apriori* 算法。

关键词：关联规则、$FP-growth$ 算法

目录

1 概述

Han[1] 等人于 2000 年提出 $FP-growth$ 算法。$FP-growth$ 算法不需要像 $Apriori$ 算法一样重复比对候选子集的支持度，只需要扫描一次数据集即可储存所有频繁模式。$FP-growth$ 算法作用于其专门的数据结构 $FP-tree$ 和 $Header-Table$，本文将分为数据结构和算法两部分进行介绍。

2 数据结构

2.1 FP-tree

$FP-tree$ 由 $FP-node$ 结点组成。$FP-node$ 的结构定义如下：

```
struct FPnode {
    string itemName;
    unsigned int frequency;
    FPnode *parent;
    vector< FPnode * > children;
};
```

由 $FP-node$ 结点组成的 $FP-tree$ 结构如下：

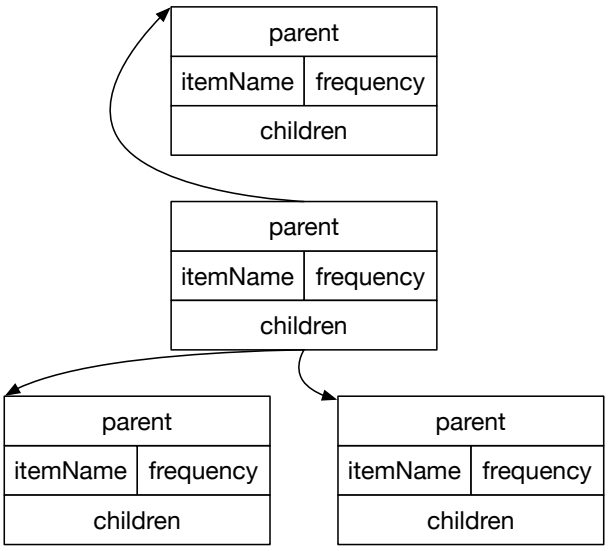


Figure 1：FP-tree 结构

2.2 Header-Table

$Header-Table$ 由 $Header-Item$ 组成。$Header-Item$ 的结构定义如下：

```
struct HeaderItem {
    string itemName;
    unsigned int frequency;
    list< FPnode * > phead;
};
```

3 构建

$FP-tree$ 和 $Header-Table$ 将同时构建，下文中将首先给出构建完成的 $FP-tree$ 和 $Header-Table$，随后给出构建二者的算法流程。

考虑如下事务集

- A B C E F O
- A C G
- E I
- A C D E G
- A C E G L
- E J
- A B C E F P
- A C D
- A C E G M
- A C E G N

设置最小支持度为 20%，则构建完成的 $FP-tree$ 和 $Header-Table$ 如下图所示（$FP-tree$ 未画出指向父结点的指针）：

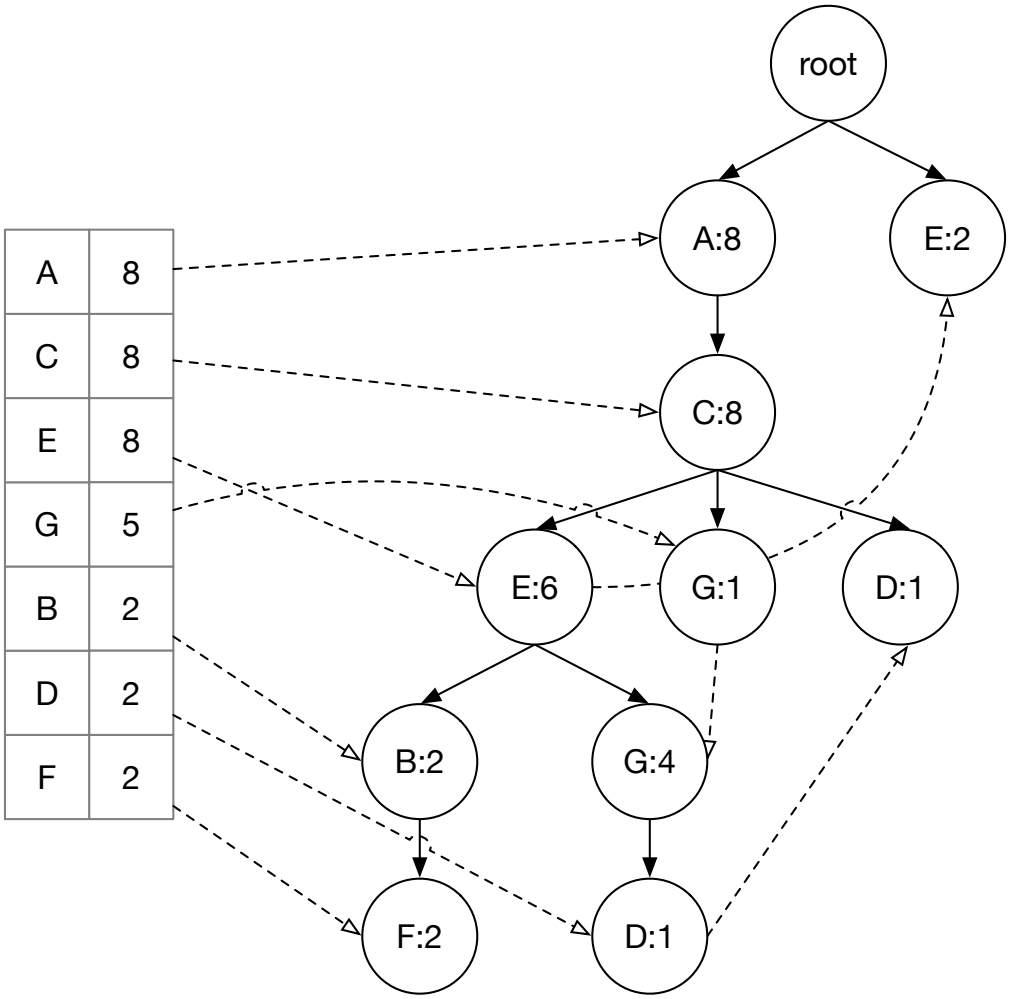


Figure 2: 构建完成的 FP-tree 和 Header Table

$FP-tree$ 满足以下特征：

- 在设定好支持度后，$FP-tree$ 中不包含任何非频繁项
- $FP-tree$ 中的任意结点值满足 $node \rightarrow frequency \leq node \rightarrow parent \rightarrow frequency$
- $FP-tree$ 中的任意一条路径代表了一个频繁模式，该频繁模式的频次由路径结尾的结点值确定。例如对于路径 $A-C-E-B$，其结尾 B 结点的值为 2，则频繁模式 $\{A, C, E, B\}$ 的频次为 2

$Header-Table$ 满足以下特征：

- $Header-Item$ 按照频次降序排列
- $phead$ 储存其在 $FP-tree$ 中的结点。例如对于 E 结点，其在 $Header-Item$ 中的频次为 8，其在 $FP-tree$ 中有两个结点，频次分别为 6 和 2

3.1 统计项目频次

Algorithm 1 Count Item Frequency

Input: transactions

Output: item_frequency

```
for transaction ∈ transactions do
    for item ∈ transaction do
        if item ∈ item_frequency then
            item_frequency[item] ← item_frequency[item]+1
        else
            item_frequency[item] ← 1
        end if
    end for
end for
return item_frequency
```

遍历事务集（$transactions$），统计所有项目的频次。

A	B	C	D	E	F	G	I	J	M	N	O	P
8	2	8	2	8	2	5	1	1	1	1	1	1

表 1: 项目频次对照表

3.2 删除不频繁的项

删除频次小于最小支持度（$threshold$）的项目。

A	B	C	D	E	F	G
8	2	8	2	8	2	5

表 2: 频繁项目频次对照表

Algorithm 2 Delete Unfrequent Item

Input: item_frequency, threshold

Output: item_frequency

```
for item ∈ item_frequency do
    if item.frequency < threshold then
        delete item
    end if
end for
return item_frequency
```

3.3 对事务集重新排序

Algorithm 3 Rerank Transactions

Input: item_frequency, transactions

Output: transactions

```
for transaction ∈ transactions do
    for item ∈ transaction do
        if item ∉ item_frequency then
            delete item
        end if
    end for
    descend_sort(transaction)
end for
return transactions
```

删除不频繁的项，并将每一个事务记录中的项目按照其频次降序排列。

- A C E B F
- A C G
- E
- A C E G D
- A C E G
- E
- A C E B F

- A C D
- A C E G
- A C E G

3.4 初始化 Header-Table

Algorithm 4 Initialize Header-Table

Input: item_frequency

Output: header_table

```
initialize header_table
for item ∈ item_frequency do
    add item to header_table
end for
descend_sort(header_table)
return header_table
```

初始化 $Header-Table$，此时每一个 $Header-Item$ 中的 $phead$ 为空。

A	C	E	G	B	D	F
8	8	8	5	2	2	2

表 3: Header-Table

3.5 构建 FP-tree

Algorithm 5 Create FP-tree

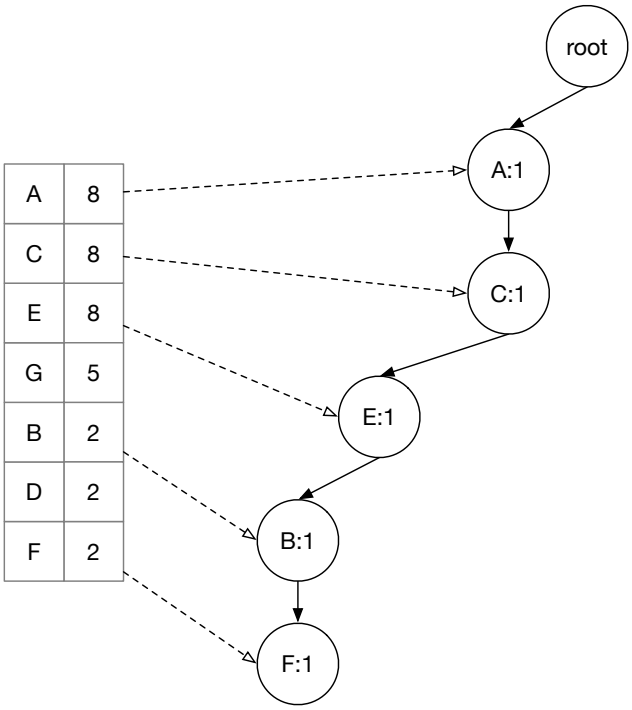
Input: transactions, header_table

Output: FP-tree_root

```
initialize FP-tree_root
for transaction ∈ transactions do
    initialize FP-node ← FP-tree_root
    j ← 0
    while j ≠ size_of(transaction) do
        initialize pos ← position of transaction[j] in FP-node → children
        if pos exists then
            FP-node → children[pos] → frequency ←
            FP-node → children[pos] → frequency+transaction[j].frequency
            FP-node ← FP-node→children[pos]
            j ← j+1
        else
            while j ≠ size_of(transaction) do
                initialize new_FP-node ← transaction[j]
                initialize pos ← position of new_FP-node.itemName in header_table
                add new_FP-node to header_table[pos].phead
                add new_FP-node to FP-node.children
                new_FP-node.parent ← FP-node
                FP-node ← new_FP-node
                j ← j+1
            end while
            break
        end if
    end while
end for
return FP-tree_root
```

构建 $FP-tree$ 的同时将 $FP-node$ 填入 $Header-Item$ 的 $phead$ 中。

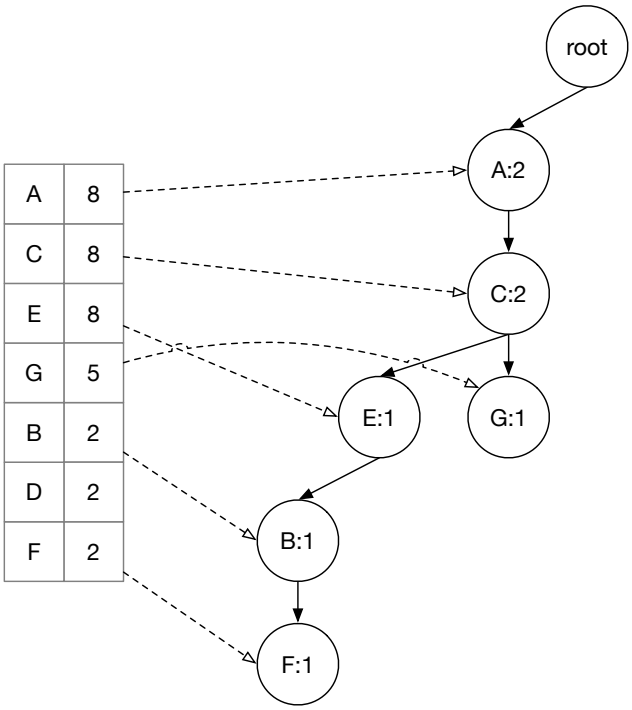
以下是对例子中前 5 条事务构建 $FP-tree$ 的过程。



A	8
C	8
E	8
G	5
B	2
D	2
F	2

A	C	E	B	F
A	C	G		
E				
A	C	E	G	D
A	C	E	G	
E				
A	C	E	B	F
A	C	D		
A	C	E	G	
A	C	E	G	

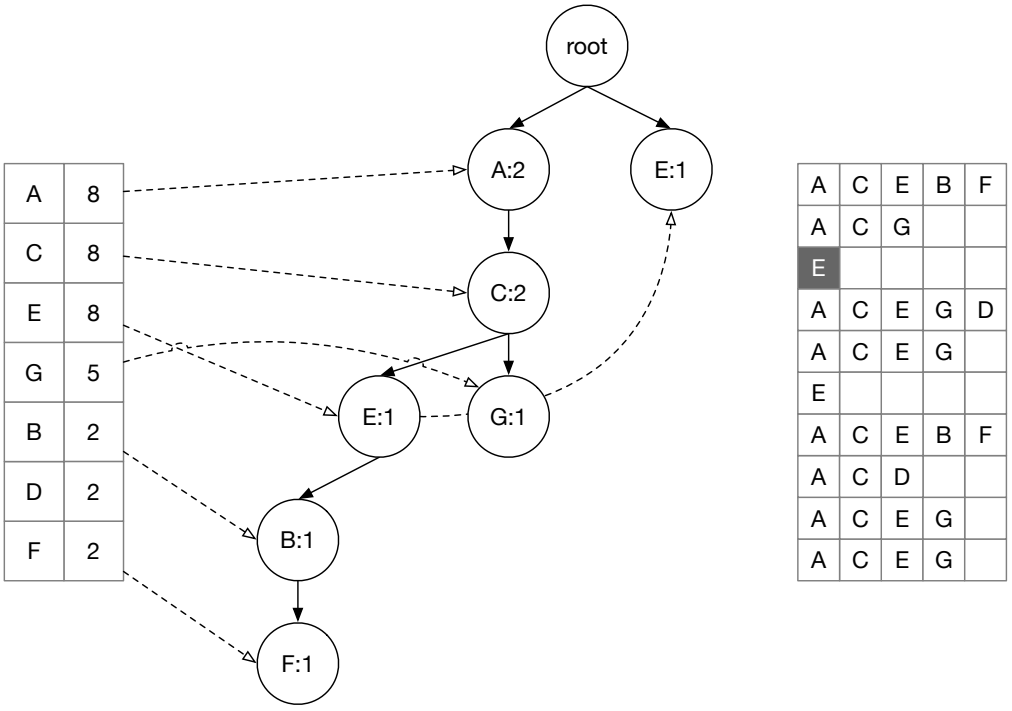
对于第一条被构建的 FP 子树（路径）而言，其每个结点都是新结点，故将形成一个新路径。



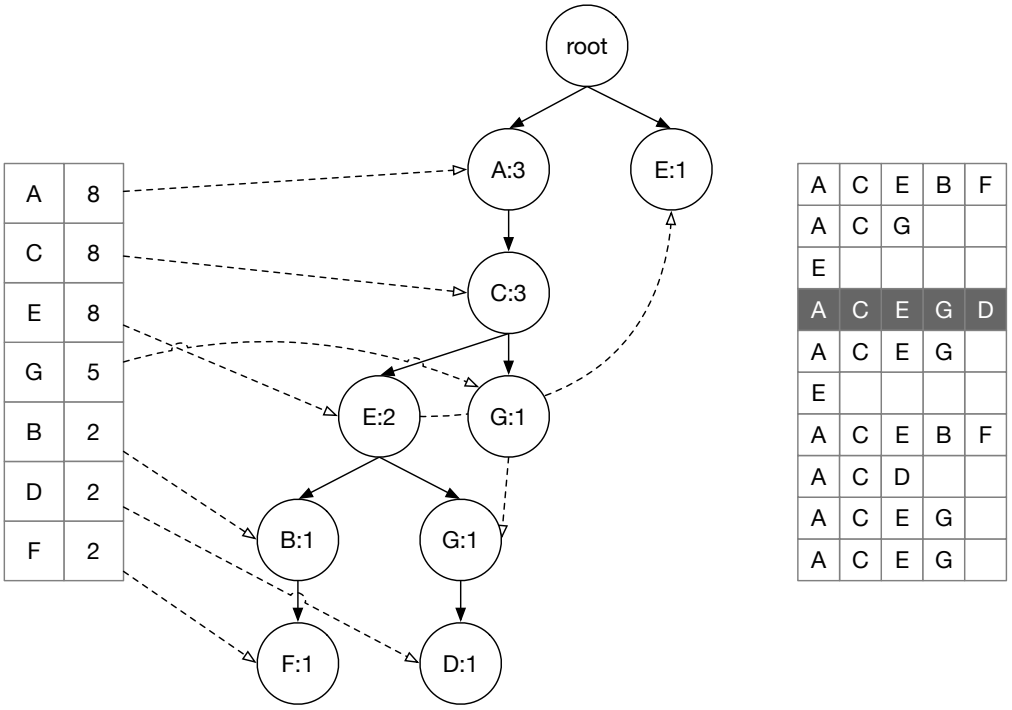
A	8
C	8
E	8
G	5
B	2
D	2
F	2

A	C	E	B	F
A	C	G		
E				
A	C	E	G	D
A	C	E	G	
E				
A	C	E	B	F
A	C	D		
A	C	E	G	
A	C	E	G	

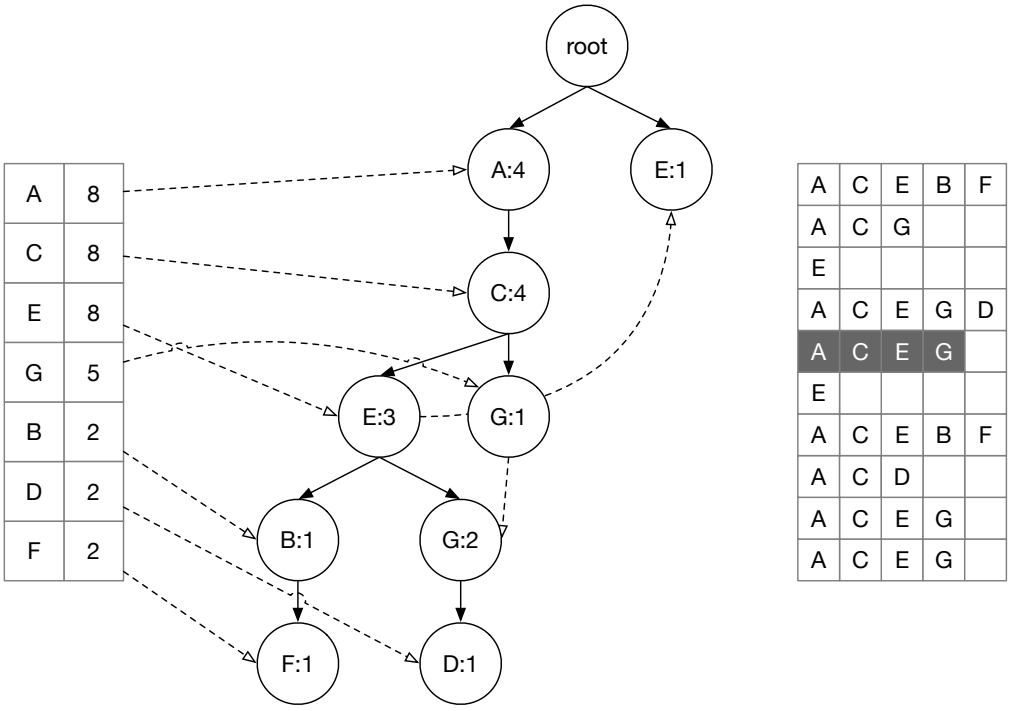
对于此条被构建的 FP 子树（路径）而言，因为其前两项 A、C 是树中已经出现的前缀，故不会出现新的分支。而最后一项 G 是新出现的项，故将形成一个新路径。



此条与上一条同理。由于 E 项在树中已经出现，故会被连接到 $Header - Item$ 的 $phead$ 中。



原理同上。


root
A:4
E:1
C:4
E:3
G:1
B:1
G:2
F:1
D:1
A 8
C 8
E 8
G 5
B 2
D 2
F 2
A C E B F
A C G
E
A C E G D
A C E G
E
A C E B F
A C D
A C E G
A C E G

原理同上。

4 挖掘

4.1 条件频繁模式

Algorithm 6 Conditional Frequent Pattern

Input: header_item, threshold

Output: item_frequency

initialize item_frequency
for FP-node ∈ header_item.phead **do**
  initialize leaf_node
  leaf_node → parent ← FP-node → parent
  initialize leaf_frequency ← FP-node → frequency
  while leaf_node is not root **do**
    leaf_node ← leaf_node → parent
    item_frequency[leaf_node → itemName] ← leaf_node → frequency
  end while
end for
delete_unfrequent_item(item_frequency, threshold)[2]
descend_sort(item_frequency)
return item_frequency

获取仅包含该 $header\_item$ 的频繁模式，以频次的降序排列。

从 $Header-Table$ 的一个 $Header-Item$ 的 $phead$ 开始，遍历其每一个项（在树中为叶子结点）。对于每一个叶子结点而言，回溯地获得其路径（直至根结点）。合并所有路径上的频次。最后降序排列。

4.2 所有频繁模式

利用深度优先搜索（BFS）获取条件频繁模式的幂集，由于条件频繁模式中的项目已经按照频次的降序排列，故只需返回最后访问的结点的频次即可。

参考文献

[1] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[J]. ACM sigmod record, 2000, 29(2): 1-12.