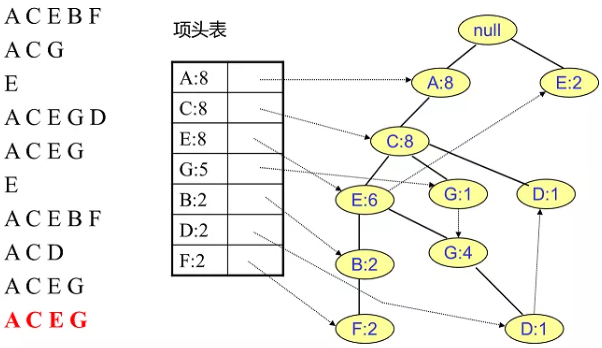
FP-growth理论与实践



Apriori和FP-Growth是两种常用的关联规则挖掘算法，Apriori算法需要多次扫描数据，I/O是很大的瓶颈。为了解决这个问题，FP Tree算法（也称FP Growth算法）采用了一些技巧，无论多少数据，只需要扫描两次数据集，因此提高了算法运行的效率。

# 1、相关概念

## 1.1 FP-Tree

就是把事务数据表中的各个事务数据项按照支持度排序后，把每个事务中的数据项按降序依次插入到一棵以NULL为根结点的树中，同时在每个结点处记录该结点出现的支持度。

## 1.2 条件模式基

包含FP-Tree中与后缀模式一起出现的前缀路径的集合。也就是同一个频繁项在FP树中的所有节点的祖先路径的集合。比如I3在FP树中一共出现了3次，其祖先路径分别是{I2，I1：2(频度为2)}，{I2：2}和{I1：2}。这3个祖先路径的集合就是频繁项I3的条件模式基。

## 1.3 条件树

将条件模式基按照FP-Tree的构造原则形成的一个新的FP-Tree。

# 2、关联规则

## 2.1 关联规则简介

关联规则是在频繁项集的基础上得到的。关联规则指由集合A，可以在某置信度下推出集合B。通俗来说，就是如果A发生了，那么B也很有可能会发生。举个例子，有关联规则如：{'鸡蛋', '面包'} -> {'牛奶'}，该规则的置信度是0.9，意味着在所有买了鸡蛋和面包的客户中，有90%的客户还买了牛奶。关联规则可以用来发现很多有趣的规律。这其中需要先阐明两个概念：支持度和置信度。

### 2.1.1 支持度Support

支持度指某频繁项集在整个数据集中的比例。假设数据集有10条记录，包含{'鸡蛋', '面包'}的有5条记录，那么{'鸡蛋', '面包'}的支持度就是5/10 = 0.5。

### 2.1.2 置信度Confidence

置信度是针对某个关联规则定义的。有关联规则如{'鸡蛋', '面包'} -> {'牛奶'}，它的置信度计算公式为{'鸡蛋', '面包', '牛奶'}的支持度/{'鸡蛋', '面包'}的支持度。假设{'鸡蛋', '面包', '牛奶'}的支持度为0.45，{'鸡蛋', '面包'}的支持度为0.5，则{'鸡蛋', '面包'} -> {'牛奶'}的置信度为 0.45 / 0.5 = 0.9。

## 2.2 关联规则原理

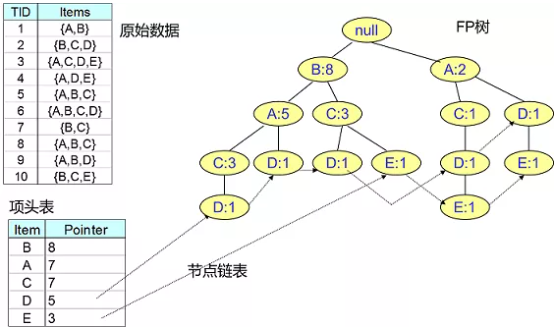
关联规则挖掘首先需要对上文得到的频繁项集构建所有可能的规则，然后对每条规则逐个计算置信度，输出置信度大于最小置信度的所有规则。以频繁项集{a,b,c}为例，构建所有可能的规则：{b,c} -> {a}, {a,c} -> {b},{a,b} -> {c},{c} -> {a,b},{b} -> {a,c},{a} -> {b,c}。对每条规则计算置信度后，输出满足要求的规则即可。

关联规则用于发现if -> then这样的规则，并可以给出这条规则的可信度（即置信度）。现实场景中可以用来发现很多规律，下面举个例子。在信息安全领域，需要根据已有流量数据制定规则，来判断是否触发安全报警。如规则{'数据包大'，'多个ip地址同时发送数据'} -> {'异常'}，该规则的置信度为0.85。这条规则表示，当流量数据包大，并有多个ip地址同时向目标ip发送数据时，则有85%的概率存在异常，需要触发报警。

# 3、FR-growth算法

## 3.1 FP Tree数据结构

为了减少I/O次数，FP Tree算法引入了一些数据结构来临时存储数据。这个数据结构包括三部分，如下图所示：



第一部分是一个项头表。里面记录了所有的1项频繁集出现的次数，按照次数降序排列。比如上图中B在所有10组数据中出现了8次，因此排在第一位，这部分好理解。第二部分是FP Tree，它将我们的原始数据集映射到了内存中的一颗FP树，这个FP树比较难理解，它是怎么建立的呢？这个我们后面再讲。第三部分是节点链表。所有项头表里的1项频繁集都是一个节点链表的头，它依次指向FP树中该1项频繁集出现的位置。这样做主要是方便项头表和FP Tree之间的联系查找和更新，也好理解。

下面我们讲项头表和FP树的建立过程。

## 3.2 项头表的建立

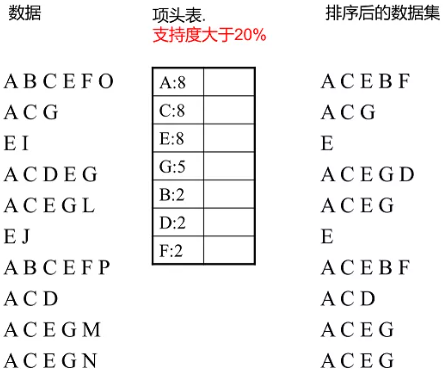
FP树的建立需要首先依赖项头表的建立。首先我们看看怎么建立项头表。

我们第一次扫描数据，得到所有频繁一项集的的计数。然后删除支持度低于阈值的项，将1项频繁集放入项头表，并按照支持度降序排列。接着第二次也是最后一次扫描数据，将读到的原始数据剔除非频繁1项集，剩下的这些元素称为频繁项，并按照支持度降序排列，建立FP树。

上面这段话很抽象，我们用下面这个例子来具体讲解。我们有10条数据，首先第一次扫描数据并对1项集计数，我们发现O，I，L，J，P，M, N都只出现一次，支持度低于20%的阈值，因此他们不会出现在下面的项头表中。剩下的A,C,E,G,B,D,F按照支持度的大小降序排列，组成了我们的项头表。

接着第二次扫描数据，对于每条数据剔除非频繁1项集，并按照支持度降序排列。比如数据项ABCEFO，里面O是非频繁1项集，因此被剔除，只剩下了ABCEF。按照支持度的顺序排序，它变成了ACEBF。其他的数据项以此类推。为什么要将原始数据集里的频繁1项数据项进行排序呢？这是为了我们后面的FP树的建立时，可以尽可能的共用祖先节点。

通过两次扫描，项头表已经建立，排序后的数据集也已经得到了，下面我们再看看怎么建立FP树。

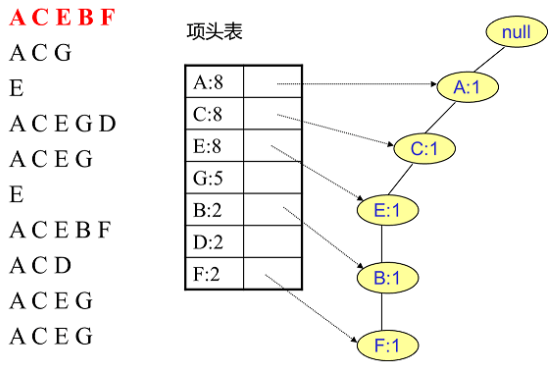


## 3.3 FP Tree的建立

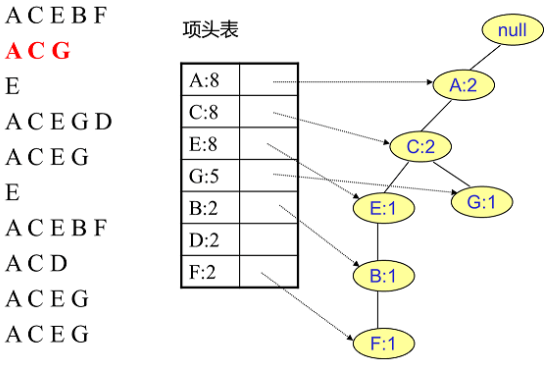
有了项头表和排序后的数据集，我们就可以开始FP树的建立了。开始时FP树没有数据，建立FP树时我们一条条的读入排序后的数据集，插入FP树，插入时按照排序后的顺序，插入FP树中，排序靠前的节点是祖先节点，而靠后的是子孙节点。如果有共用的祖先，则对应的公用祖先节点计数加1。插入后，如果有新节点出现，则项头表对应的节点会通过节点链表链接上新节点。直到所有的数据都插入到FP树后，FP树的建立完成。

似乎也很抽象，我们还是用第二节的例子来描述。

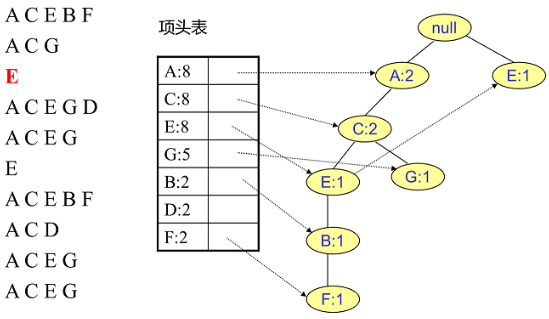
首先，我们插入第一条数据ACEBF，如下图所示。此时FP树没有节点，因此ACEBF是一个独立的路径，所有节点计数为1，项头表通过节点链表链接上对应的新增节点。

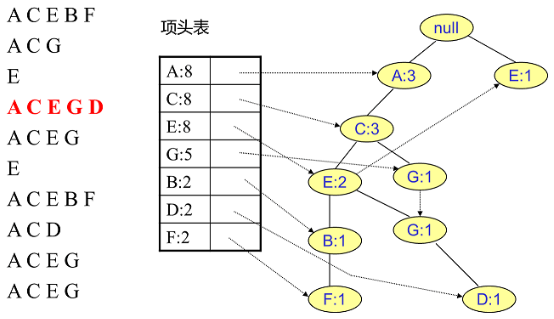


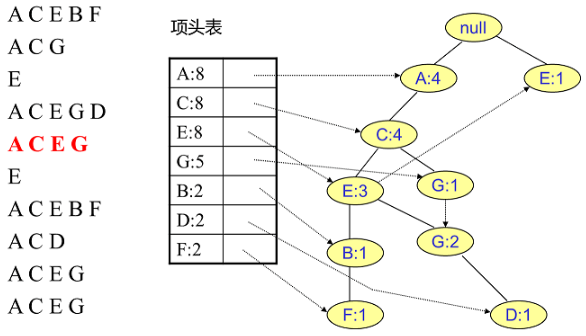
接着我们插入数据ACG，如下图所示。由于ACG和现有的FP树可以有共有的祖先节点序列AC，因此只需要增加一个新节点G，将新节点G的计数记为1。同时A和C的计数加1成为2。当然，对应的G节点的节点链表要更新。

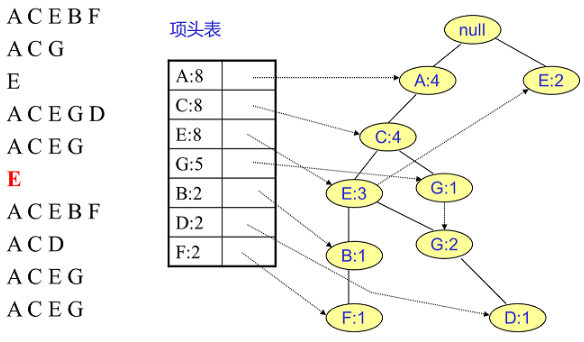


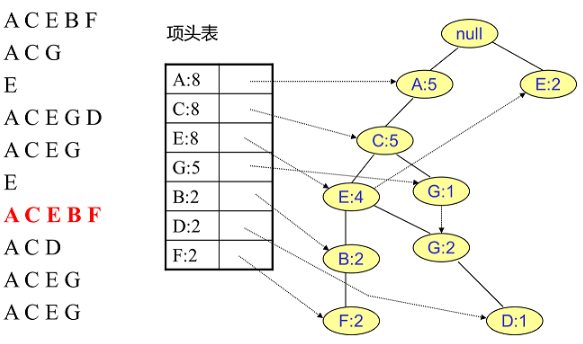
同样的办法可以更新后面8条数据，如下8张图。由于原理类似，这里就不多文字讲解了，大家可以自己去尝试插入并进行理解对比。相信如果大家自己可以独立的插入这10条数据，那么FP树建立的过程就没有什么难度了。

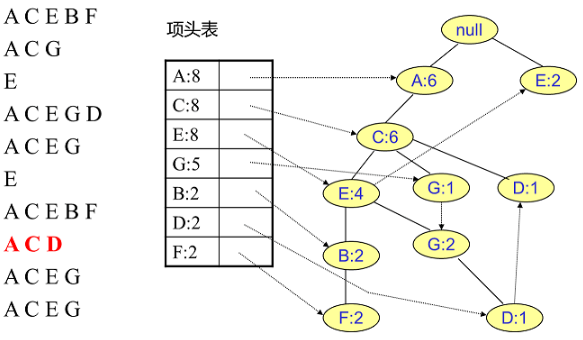


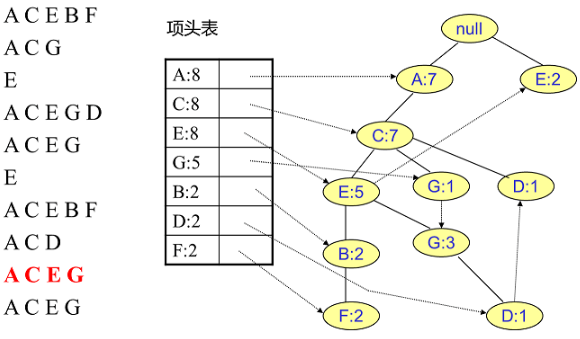


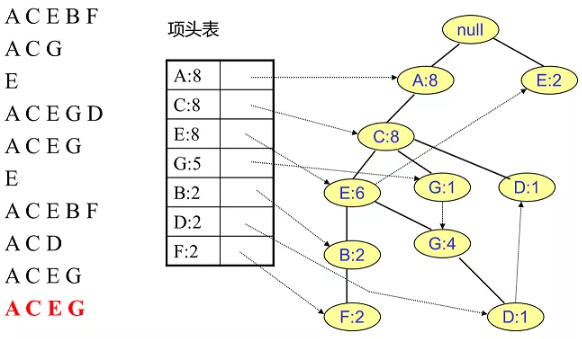








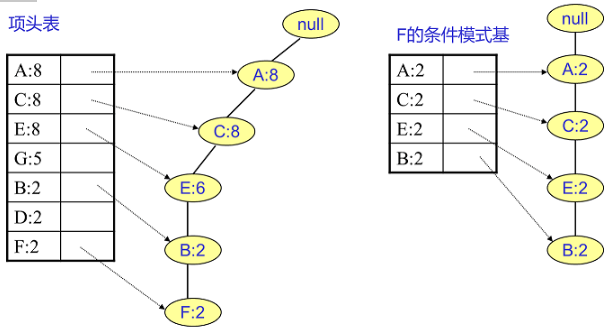




## 3.4 FP Tree挖掘频繁项集

FP树建立起来了，那么怎么去挖掘频繁项集呢？下面我们讲如何从FP树里挖掘频繁项集。得到了FP树和项头表以及节点链表之后，需要对每一个频繁项，逐个挖掘频繁项集。首先要从项头表的底部项依次向上挖掘。对于项头表对应于FP树的每一项，我们要找到它的条件模式基（也称条件FP树）。所谓条件模式基是以当前要挖掘的节点作为叶子节点所对应的FP子树。得到这个FP子树，将子树中每个节点的的计数设置为叶子节点的计数，并删除计数低于支持度的节点。从这个条件模式基，就可以递归挖掘得到频繁项集了。

还是以上面的例子来讲解。我们看看先从最底下的F节点开始，我们先来寻找F节点的条件模式基，由于F在FP树中只有一个节点，因此候选就只有下图左所示的一条路径，对应{A:8,C:8,E:6,B:2, F:2}。我们接着将所有的祖先节点计数设置为叶子节点的计数，即FP子树变成{A:2,C:2,E:2,B:2, F:2}。一般的，条件模式基可以不写叶子节点，因此最终的F的条件模式基如下图右所示。



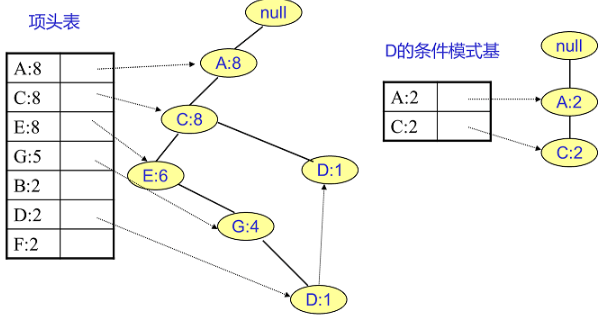
注意：此时头指针表中包含四个元素，所以对每个元素，需要获得前缀路径，并将前缀路径创建成条件FP树，直到条件FP树中只包含一个元素时返回。

对元素A，获得前缀路径为{}，则频繁项集返回{A,F};

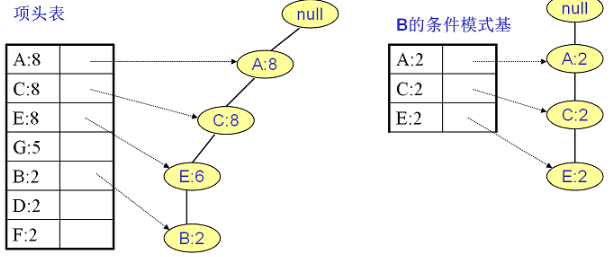
对元素B，获得前缀路径{A,C,E}，则将前缀路径创建成条件FP树。就这样递归迭代，直到FP树中只有一个节点。

就这样，很容易得到F的频繁2项集为{A:2,F:2}, {C:2,F:2}, {E:2,F:2}, {B:2,F:2}。递归合并二项集，得到频繁三项集为{A:2,C:2,F:2}，{A:2,E:2,F:2},...还有一些频繁三项集，就不写了。当然一直递归下去，最大的频繁项集为频繁5项集，为{A:2,C:2,E:2,B:2,F:2}

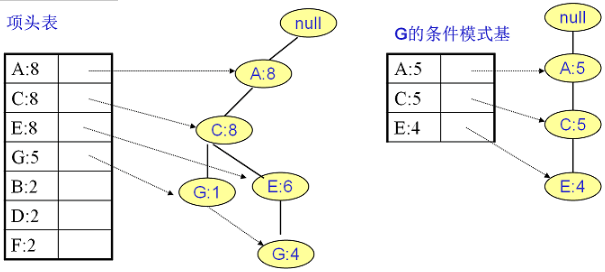
F挖掘完了，我们开始挖掘D节点。D节点比F节点复杂一些，因为它有两个叶子节点，因此首先得到的FP子树如下图左。我们接着将所有的祖先节点计数设置为叶子节点的计数，即变成{A:2, C:2,E:1 G:1,D:1, D:1}，此时E节点和G节点由于在条件模式基里面的支持度低于阈值，被我们删除，最终在去除低支持度节点并不包括叶子节点后D的条件模式基为{A:2, C:2}。通过它，我们很容易得到D的频繁2项集为{A:2,D:2}, {C:2,D:2}。递归合并二项集，得到频繁三项集为{A:2,C:2,D:2}。D对应的最大的频繁项集为频繁3项集。



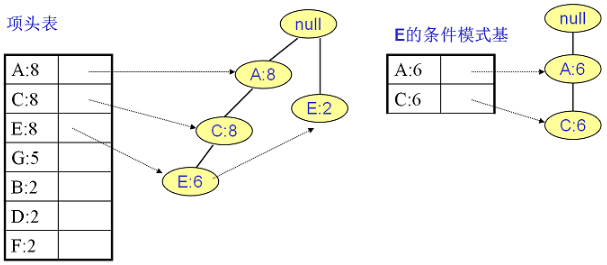
同样的方法可以得到B的条件模式基如下图右边，递归挖掘到B的最大频繁项集为频繁4项集{A:2, C:2, E:2,B:2}。



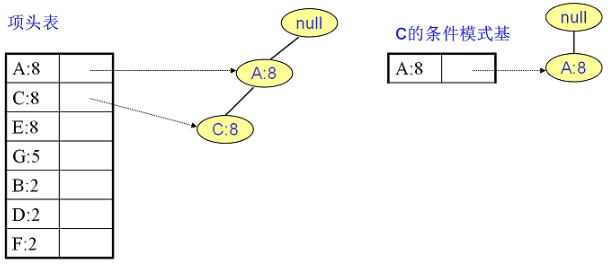
继续挖掘G的频繁项集，挖掘到的G的条件模式基如下图右边，递归挖掘到G的最大频繁项集为频繁4项集{A:5, C:5, E:4,G:4}。



E的条件模式基如下图右边，递归挖掘到E的最大频繁项集为频繁3项集{A:6, C:6, E:6}。



C的条件模式基如下图右边，递归挖掘到C的最大频繁项集为频繁2项集{A:8, C:8}。



至于A，由于它的条件模式基为空，因此可以不用去挖掘了。

至此我们得到了所有的频繁项集，如果我们只是要最大的频繁K项集，从上面的分析可以看到，最大的频繁项集为5项集。包括{A:2, C:2, E:2,B:2,F:2}。

通过上面的流程，相信大家对FP Tree的挖掘频繁项集的过程也很熟悉了。

## 3.5 FP Tree算法归纳

这里我们对FP Tree算法流程做一个归纳。FP Tree算法包括三步：

1）扫描数据，得到所有频繁一项集的的计数。然后删除支持度低于阈值的项，将1项频繁集放入项头表，并按照支持度降序排列。

2）扫描数据，将读到的原始数据剔除非频繁1项集，并按照支持度降序排列。

3）读入排序后的数据集，插入FP树，插入时按照排序后的顺序，插入FP树中，排序靠前的节点是祖先节点，而靠后的是子孙节点。如果有共用的祖先，则对应的公用祖先节点计数加1。插入后，如果有新节点出现，则项头表对应的节点会通过节点链表链接上新节点。直到所有的数据都插入到FP树后，FP树的建立完成。

4）从项头表的底部项依次向上找到项头表项对应的条件模式基。从条件模式基递归挖掘得到项头表项项的频繁项集。

5）如果不限制频繁项集的项数，则返回步骤4所有的频繁项集，否则只返回满足项数要求的频繁项集。

## 3.6 FP-Growth算法优缺点

优点：一般快于Apriori。

缺点：实现比较困难，在某些数据上性能下降。

# 4、FP-Growth应用

* 推荐。分析用户共同购买的商品，用于推荐
* 用于制定营销策略。如同啤酒与尿布的例子，超市如果将啤酒和尿布放在相邻的位置，会增加两者的销量。还可用于制定打折促销活动，给买了啤酒和尿布的客户打折，也可以增加销量。
* 用于发现共现词。这种场景其实我们经常会遇到。当我们在浏览器中输入"频繁项集"时，浏览器自动弹出如"频繁项集 置信度"，"频繁项集 关联规则"等备选记录，我们每每都会感叹浏览器的智能，其实这里的秘诀就是频繁项集。也就是说，在大量的用户搜索记录中，"频繁项集"和"置信度"共同出现在了大多数的搜索记录中。同理，"频繁项集"和"关联规则"也频繁得共同出现在搜索记录中。
* 用于发现事物的热点信息。从新闻报道和微博中获取关于某事物的相关文档，然后应用频繁项集挖掘算法可以得到该事物的热点新闻。

# 5、实践

参考：

https://github.com/jpegbert/MachineLearning/tree/master/fp\_growth

包含python库实现，python实现和pyspark实现

# 6、总结

FP-growth是目前业界经典的频繁项集和关联规则挖掘的算法。相比于 Apriori 模型，FP-growth 模型只需要扫描数据库两次，巧妙的利用了树结构，极大得减少了数据读取次数并显著得提升了算法效率。

在实践中，FP Tree算法是可以用于生产环境的关联算法，而Apriori算法则做为先驱，起着关联算法指明灯的作用。

参考：

<https://mp.weixin.qq.com/s/k9iNioYvXpUH6Mc6V-Ltlg>

<https://mp.weixin.qq.com/s/Bs2v9aPLbl9JhAn_r-jisw>

<https://mp.weixin.qq.com/s/k9iNioYvXpUH6Mc6V-Ltlg>

<https://mp.weixin.qq.com/s/TOAGn1pttgMFcCzM7dVJbA>

<https://mp.weixin.qq.com/s/XDu8Ij8a0xT4k93JiOqM3w>

<https://mp.weixin.qq.com/s/IfJI5lBtuJFKwBPLSLbUFA>

<https://mp.weixin.qq.com/s/ahmVB0ktJ2PG37ErO-vIiQ>

<https://mp.weixin.qq.com/s/Pm2FqVYP7uk_uAbAI_rm6Q>

<https://mp.weixin.qq.com/s/XGFW-z4kEMh8CVRzGf5ijw>