In [2]:

```
import numpy as np
from collections import OrderedDict
import csv

# FP tree stucture
class TreeNode:
    def __init__(self, name, freq, parent):
        self.name = name #Name of the item
        self.freq = freq # Frequency of the item
        self.parent = parent # Parent node
        self.child = OrderedDict() # contains all the children node information
        self.link = None # for linking to nodes with same name

    # Method to display the FP-Tree or conditional FP-Tree as a nested list
    def displayTreeList(self):
        print(self.name, self.freq,end='')
        if len(self.child)>0:
            print(",[",end='')
        for c in self.child.values():
            print("[",end='')
            c.displayTreeList()
            if len(c.child)==0:
                print("]",end='')
        print("]",end='')


"""Writes the frequent itemsets to a CSV file"""
def exportToFile(data):
    with open(output_file_name, "w",  newline='') as f:
        writer = csv.writer(f, delimiter=',')
        for row in data:
            writer.writerows([[row]])


"""The most recent node is linked to the previous node with same name."""
def similarItemTableUpdate(similarItem, presentNode):
    while (similarItem.link != None):
        similarItem = similarItem.link
    similarItem.link = presentNode


#Take dataset as input, gives dictionary of items which satisfies support
def fpTreePreprocess(docName, threshold):
    data = np.genfromtxt(docName, delimiter=fileDelimiter, dtype=str)
    itemFreq = {}
    for (x,y), value in np.ndenumerate(data):
        # Check if the item is a null value or not.
        # If not, append it to the  itemFreq dictionary.
        if value != '':
            if value not in itemFreq:
                itemFreq[value] = 1
            else:
                itemFreq[value] += 1
    # Removing items whcih are below the given support value
    itemFreq = {k:v for k,v in itemFreq.items() if v >= threshold}
    return data, itemFreq


# second scan on input received after first scan
def fpTreeReorder(data, itemFreq):
    root = TreeNode('Root',1,None)
    #Sort the frequent item dictionary based on the frequency of the items
    #If two items have the same frequency, the keys are arranged alphabetically
    sortedItemFreq = sorted(itemFreq.items(), key=lambda x: (-x[1],x[0]))
    # The similar item table is also created with all the frequent items.
    sortedKeys = []
    similarItemDict = {}
    for key in sortedItemFreq:
        similarItemDict[key[0]] = None # Initially all the values are 'None'
        sortedKeys.append(key[0]) # A list of the sorted item structure
    # 2nd scan of the database.
    for row in data:
        # Deletes any item whose frequency is not above the minimum support
        trans = []
        for col in row:
            if col in itemFreq:
                trans.append(col)
        # Orders the items in a transaction based on its frequency.
        orderedTrans = []
        for item in sortedKeys:
            if item in trans:
                orderedTrans.append(item)
        # Once ordered, the transaction is sent to be updated in the FP-Tree
        if len(orderedTrans)!= 0:
            fpTreeCreateAndUpdate(root, orderedTrans, similarItemDict)
    return root, similarItemDict


"""This function recursively creates the FP-Tree for each transaction."""
def fpTreeCreateAndUpdate(initNode, trans, similarItemDict):
    # If the child is already present, increment its count
    if trans[0] in initNode.child:
        initNode.child[trans[0]].freq += 1
    # Else, create a new node for the child and link it to its parent
    else:
        initNode.child[trans[0]] = TreeNode(trans[0], 1, initNode)
        # For every newly created node, the Similar-Item table is updated
        if similarItemDict[trans[0]] == None:
            # For the 1st node, replace the 'None' value with the node
            similarItemDict[trans[0]] = initNode.child[trans[0]]
        else:
            # Traverse till the last similar node, and update the new node
            similarItemTableUpdate(similarItemDict[trans[0]],\
                                      initNode.child[trans[0]])
    # The function is recursively called for every item in a transaction
    if len(trans) > 1:
        fpTreeCreateAndUpdate(initNode.child[trans[0]], trans[1::],\
                                  similarItemDict)


"""Function to create the FP-Tree for every frequent occuring item
in the main FP-Tree.
The function works exactly similar to the fpTreeCreateAndUpdate() function,
except here the similar-item table is not updated"""
def conditionalFpTree(name,initNode,data):
    if len(data) > 0 and data[0][0] == name:
        # Skip the conditional tree if no extra frequent items are occuring
        if len(data)>1:
            conditionalFpTree(name,initNode,data[1::])
    if len(data) > 0 and data[0][0] != name:
        # If the item is present as a child, increment its count
        if data[0][0] in initNode.child:
            initNode.child[data[0][0]].freq += data[0][1]
        # Else, create a new child node and update its frequency
        else:
            initNode.child[data[0][0]] = TreeNode(data[0][0],data[0][1],\
                           initNode)
        # Continue to recursively create the conditional FP-Tree for each item
        if len(data) >1:
            conditionalFpTree(name,initNode.child[data[0][0]],data[1::])

"""Function to create the FP-Tree for every time present in the
Similar-Item Table. Each freqently occuring itemset above the threshold is also cons
"""
def createLeafCondBase(similarItemDict, threshold):
    finalCondBase = []
    # Go through every key-value pair present in the Similar-Item Table
    for key,value in similarItemDict.items():
        finalCondBase_key = []
        conditionBase = []
        leafItemFreq = OrderedDict()
        # Within each key, traverse through every linked node value till end
        while value != None:
            path = []
            leafNode = value
            leafFreq = value.freq
            #Within each node, traverse till the parent node and append details
            while leafNode.parent != None:
                leafDetails = [leafNode.name, leafFreq]
                path.append(leafDetails) # append the name and value
                leafNode = leafNode.parent # Go to the parent of that node
            # Insert the whole path to conditionBase
            conditionBase.insert(0,path)
            # Once the  particular node is finished, increment to value.link
            # Then you can traverse for the next same-name node
            value = value.link
        # A frequent item-set dictionary is created for every leaf node
        for row in conditionBase:
            for col in row:
                if col[0] not in leafItemFreq:
                    leafItemFreq[col[0]] = col[1]
                else:
                    leafItemFreq[col[0]] += col[1]
        #Items below threshold are removed before creating the conditional base
        leafItemFreq = {k:v for k,v in leafItemFreq.items() \
                          if v >= threshold}
        # For every transaction in the conditionBase, the items are stored
        for row in conditionBase:
            temp = []
            temp_tree = []
            for col in row:
                if col[0] in leafItemFreq:
                    temp.append(col[0]) # stores only the name of the item
                    temp_tree.append(col) # stores both name and frequency
            #Contains all the frequent items for a particular conditional leaf
            finalCondBase.append(temp) # used for question 2A
            finalCondBase_key.append(temp_tree) # used for question 2B
        ## Starting code for question 2B
        condLeaf = key
        cond_root = TreeNode('Null Set',1,None)
        # Creates the conditional tree from the above conditonal pattern base
        for row in finalCondBase_key:
            conditionalFpTree(condLeaf,cond_root,row)
        # Prints the conditional tree in nested list format if height > 1
        if len(cond_root.child) > 1:
            print('\n--------',"Conditional FP-Tree for",condLeaf,'--------')
            print("[",end='')
            cond_root.displayTreeList()
            print('\n')
        ## Ending code for question 2B
    #Removes any duplicate rows having the same itemset
    unique_cond_base_set = set(map(tuple,finalCondBase))
    unique_cond_base_list = list(unique_cond_base_set)
    unique_cond_base = map(list,unique_cond_base_list)
    exportToFile(unique_cond_base) # Exports the asnwer to a CSV file


"""Main part of the code"""
# Required User Inputs
support = 500
file_name = 'transactions.csv'
fileDelimiter = ','
output_file_name = "output.csv"
# Function calling in the main program to implement FP Growth algorithm
dataset, freq_items = fpTreePreprocess(file_name, support)
fptree_root, header_table = fpTreeReorder(dataset, freq_items)
createLeafCondBase(header_table,support)
```

```
-------- Conditional FP-Tree for 22383 --------
[Null Set 1,[[20727 587,[[20725 361]]][20725 302]]]


-------- Conditional FP-Tree for 21931 --------
[Null Set 1,[[22386 518,[[85099B 416]]][85099B 317]]]


-------- Conditional FP-Tree for 22411 --------
[Null Set 1,[[21931 525,[[85099B 391]]][85099B 292]]]


-------- Conditional FP-Tree for 20728 --------
[Null Set 1,[[20727 211,[[20725 94]]][22383 544,[[20725 111]][20727 315,[[20725 23
0]]]][20725 127]]]


-------- Conditional FP-Tree for 22382 --------
[Null Set 1,[[20727 200,[[20725 102]]][22383 537,[[20725 116]][20727 313,[[20725 21
6]]]][20725 129]]]


-------- Conditional FP-Tree for 22384 --------
[Null Set 1,[[20728 507,[[20727 311,[[20725 225]]][20725 100]]][20727 236,[[20725 13
7]]][20725 151]]]


-------- Conditional FP-Tree for 22697 --------
[Null Set 1,[[22699 784,[[22423 412]]][22423 106]]]


-------- Conditional FP-Tree for 22698 --------
[Null Set 1,[[22697 644,[[22699 549]]][22699 65]]]
```

In [ ]: