```
class treeNode:
  #Constructor
  def __init__(self, nameValue, numOccur, parentNode):
    self.name = nameValue
    self.count = numOccur
    self.nodeLink = None
    self.parent = parentNode
    self.children = {}

  # To Increment Count
  def inc(self, numOccur):
    self.count += numOccur

  # To Display Tree
  def display(self, ind = 1):
    print (' | '*ind,'|' ,self.name, ':', self.count)
    for child in self.children.values():
      child.display(ind + 1)


# To Create Tree
def createTree(dataset): # will create tree
  headerTable = {}

  for trans in dataset:
    for item in trans:
      headerTable[item] = headerTable.get(item, 0) + dataset[trans]

  freqItemset = set(headerTable.keys())

  if len(freqItemset) == 0:
    return None, None

  for k in headerTable:
    headerTable[k] = [headerTable[k], None]

  retTree = treeNode('NULL', 1, None)

  for tranSet, count in dataset.items():
    localD = {}
    for item in tranSet:
      if item in freqItemset:
        localD[item] = headerTable[item][0]

    if len(localD) > 0:
      orderedItems = [v[0] for v in sorted(localD.items(), key=lambda p: p[1], reverse=Tru
      updateTree(orderedItems, retTree, headerTable, count)
  return retTree, headerTable


# To Update Headers
def updateHeader(node, target):
  while(node.nodeLink != None):
    node = node.nodeLink
```

```
    node.nodeLink = target


  # To Update Tree
  def updateTree(items, inTree, headerTable, count):

    if items[0] in inTree.children:
      inTree.children[items[0]].inc(count)
    else:
      inTree.children[items[0]] = treeNode(items[0], count, inTree)

      if headerTable[items[0]][1] == None:
        headerTable[items[0]][1] = inTree.children[items[0]]
      else:
        updateHeader(headerTable[items[0]][1], inTree.children[items[0]])

    if len(items) > 1:
      updateTree(items[1::], inTree.children[items[0]], headerTable, count)


  # To Ascend Tree
  def ascendTree(leaf, raw):
    if leaf.parent != None:
      raw.append(leaf.name)
      ascendTree(leaf.parent, raw)


  data = [
          ['I1','I2','I5'],
          ['I2','I4'],
          ['I2','I3'],
          ['I1','I2','I4'],
          ['I1','I3'],
          ['I2','I3'],
          ['I1','I3'],
          ['I1','I2','I3','I5'],
          ['I1','I2','I3']
  ]

  dic = {}

  for d in data:
    tup = tuple(d)
    dic[tup] = 0

  for d in data:
    tup = tuple(d)

    if dic[tup] >= 1:
      dic[tup] += 1
    else:
      dic[tup] = 1

  ordered = {}

  for i in data:
```

```
    for j in i:
      ordered[j] = 0

  for i in data:
    for j in i:
      if ordered[j] >= 1:
        ordered[j] += 1
      else:
        ordered[j] = 1

  ordered = dict(sorted(ordered.items(), key = lambda item: (item[1],item[0]), reverse = Tru
  order = list(ordered.keys())


  # TO find Conditional Base
  def conditionalBase(base, treeNode):
    cond = {}

    while treeNode != None:
      path = []
      ascendTree(treeNode, path)

      if len(path) > 1:
        temp = path[1:]
        temp.sort(key = lambda x: order.index(x), reverse = True)
        cond[tuple(temp)] = treeNode.count
      treeNode = treeNode.nodeLink
    return cond


  rev_order = order
  rev_order.reverse()

  print("---> FP Growth Tree <---\n")
  FPtree, headertab = createTree(dic)
  FPtree.display()
  print()

  print("---> Conditional Pattern Base <---\n")

  for rev in rev_order:
    print(rev," | ",conditionalBase(rev, headertab[rev][1]))
    print('----|-------------------------------------------------')
```

```
     ---> FP Growth Tree <---

      |  | NULL : 1
      |  |  | I2 : 7
      |  |  |  | I1 : 4
      |  |  |  |  | I5 : 1
      |  |  |  |  | I4 : 1
      |  |  |  |  | I3 : 2
      |  |  |  |  |  | I5 : 1
      |  |  |  | I4 : 1
      |  |  |  | I3 : 2
      |  |  | I1 : 2
      |  |  |  | I3 : 2
```

```
---> Conditional Pattern Base <---

I4  |  {('I2',): 1, ('I2', 'I1'): 1}
----|-------------------------------------------------
I5  |  {('I2', 'I1'): 1, ('I2', 'I3', 'I1'): 1}
----|-------------------------------------------------
I1  |  {('I2',): 4}
----|-------------------------------------------------
I3  |  {('I2',): 2, ('I1',): 2, ('I2', 'I1'): 2}
----|-------------------------------------------------
I2  |  {}
----|-------------------------------------------------
```

✓ 0s completed at 23:41