Big-Data Analytics

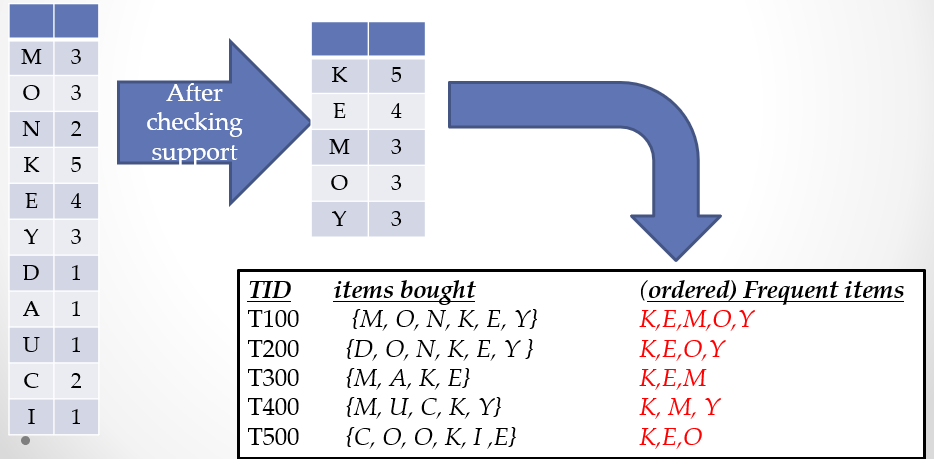
**SH-2**

Q. 1 A database has five transactions. Let min\_sup =60% and min\_conf =80%.

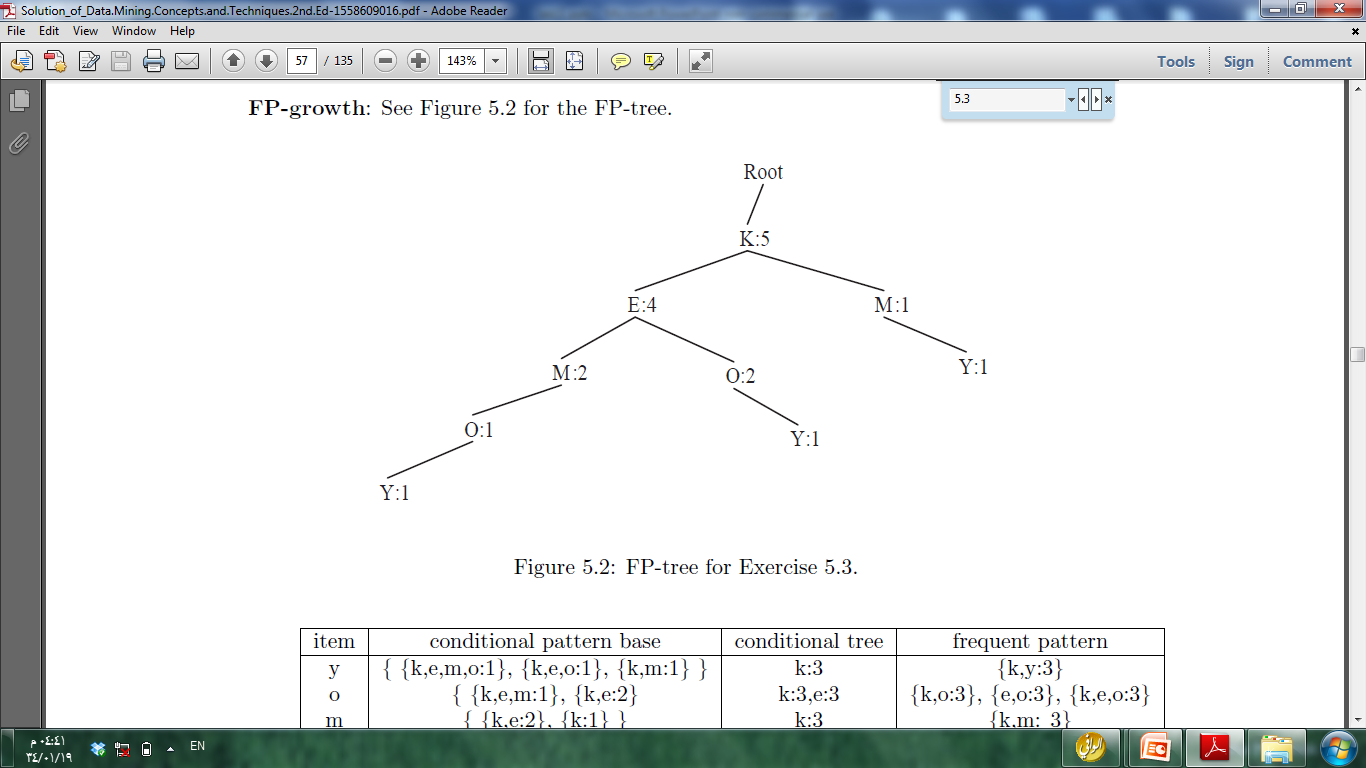
|  |  |
| --- | --- |
| TID | items bought |
| T100 | {M, O, N, K, E, Y} |
| T200 | {D, O, N, K, E, Y} |
| T300 | {M, A, K, E} |
| T400 | {M, U, C, K, Y} |
| T500 | {C, O, O, K, I, E} |

1. Find all frequent itemsets using Apriori Alogrithm.

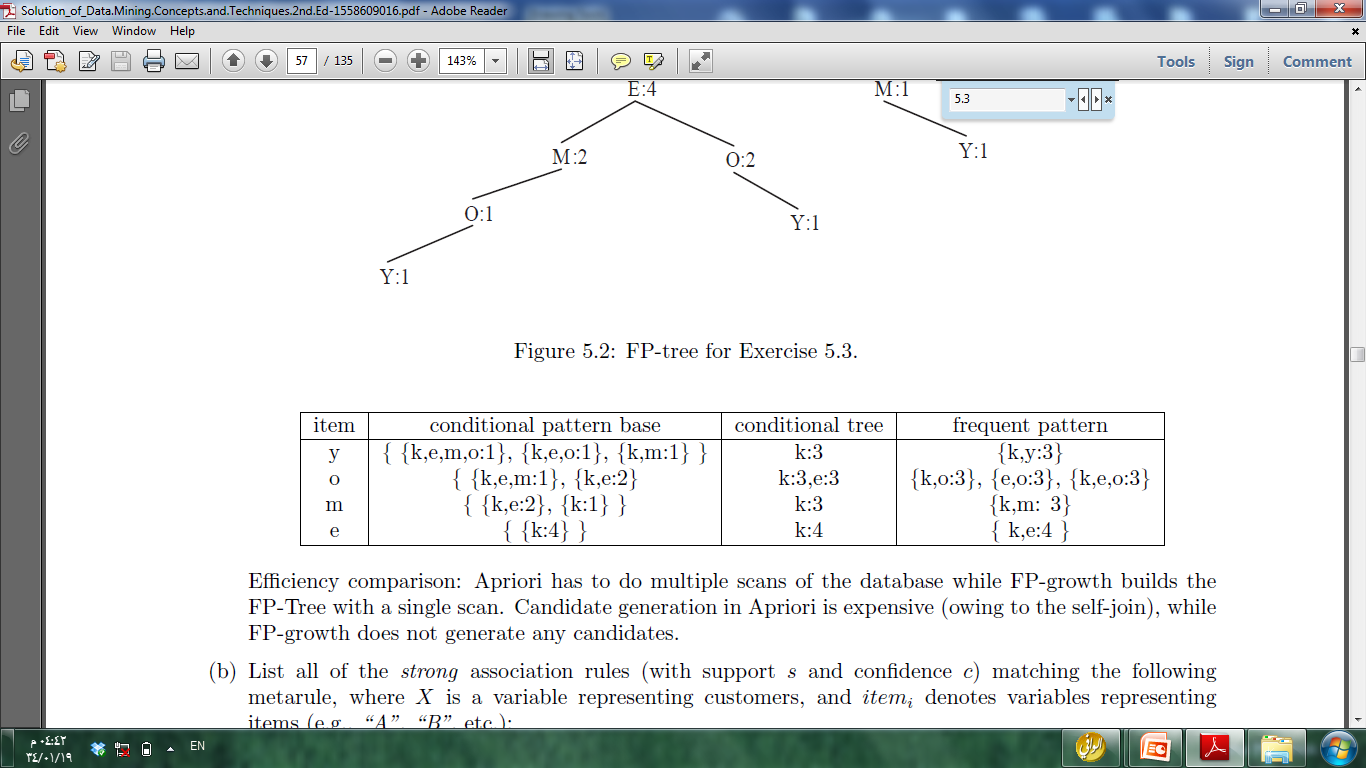
Scan DB once, find frequent 1-itemset (single item pattern) their support => 3



Generate FB-tree



Generate FB-tree – order table



1. List all the strong association rules (with support s and

confidence c).

* buys(X,k) Λ buys(X,o) => buys(X, e) [60%,100%]
* buys(X,e) Λ buys(X,o) => buys(X, k) [60%,100%]

Q2. Using a programming language that you are familiar with, such as C++ or Java or Python, implement Apriori frequent itemset mining algorithm.

import argparse

from itertools import chain, combinations

def joinset(itemset, k):

return set([i.union(j) for i in itemset for j in itemset if len(i.union(j)) == k])

def subsets(itemset):

return chain(\*[combinations(itemset, i + 1) for i, a in enumerate(itemset)])

def itemset\_from\_data(data):

itemset = set()

transaction\_list = list()

for row in data:

transaction\_list.append(frozenset(row))

for item in row:

if item:

itemset.add(frozenset([item]))

return itemset, transaction\_list

def itemset\_support(transaction\_list, itemset, min\_support=0):

len\_transaction\_list = len(transaction\_list)

l = [

(item, float(sum(1 for row in transaction\_list if item.issubset(row)))/len\_transaction\_list)

for item in itemset

]

return dict([(item, support) for item, support in l if support >= min\_support])

def freq\_itemset(transaction\_list, c\_itemset, min\_support):

f\_itemset = dict()

k = 1

while True:

if k > 1:

c\_itemset = joinset(l\_itemset, k)

l\_itemset = itemset\_support(transaction\_list, c\_itemset, min\_support)

if not l\_itemset:

break

f\_itemset.update(l\_itemset)

k += 1

return f\_itemset

def apriori(data, min\_support, min\_confidence):

# Get first itemset and transactions

itemset, transaction\_list = itemset\_from\_data(data)

# Get the frequent itemset

f\_itemset = freq\_itemset(transaction\_list, itemset, min\_support)

# Association rules

rules = list()

for item, support in f\_itemset.items():

if len(item) > 1:

for A in subsets(item):

B = item.difference(A)

if B:

A = frozenset(A)

AB = A | B

confidence = float(f\_itemset[AB]) / f\_itemset[A]

if confidence >= min\_confidence:

rules.append((A, B, confidence))

return rules, f\_itemset

def print\_report(rules, f\_itemset):

print('--Frequent Itemset--')

for item, support in sorted(f\_itemset.items(), key=lambda (item, support): support):

print('[I] {} : {}'.format(tuple(item), round(support, 4)))

print('')

print('--Rules--')

for A, B, confidence in sorted(rules, key=lambda (A, B, confidence): confidence):

print('[R] {} => {} : {}'.format(tuple(A), tuple(B), round(confidence, 4)))

def data\_from\_csv(filename):

f = open(filename, 'rU')

for l in f:

row = map(str.strip, l.split(','))

yield row

def parse\_options():

optparser = argparse.ArgumentParser(description='Apriori Algorithm.')

optparser.add\_argument(

'-f', '--input\_file',

dest='filename',

help='filename containing csv',

required=True

)

optparser.add\_argument(

'-s', '--min\_support',

dest='min\_support',

help='minimum support',

default=0.25,

type=float

)

optparser.add\_argument(

'-c', '--min\_confidence',

dest='min\_confidence',

help='minimum confidence',

default=0.5,

type=float

)

return optparser.parse\_args()

def main():

options = parse\_options()

data = data\_from\_csv(options.filename)

rules, itemset = apriori(data, options.min\_support, options.min\_confidence)

print\_report(rules, itemset)

if \_\_name\_\_ == '\_\_main\_\_':

main()