Hands-on Exercise for FPM Module

1. Exploring properties of the dataset accidents_10k.dat. Read more about it here: http://fimi.uantwerpen.be/data/accidents.pdf (http://fimi.uantwerpen.be/data/accidents.pdf)

In [1]:

```
!head accidents_10k.dat
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31
2 5 7 8 9 10 12 13 14 15 16 17 18 20 22 23 24 25 27 28 29 32 33 34 35 36 3
7 38 39
7 10 12 13 14 15 16 17 18 20 25 28 29 30 33 40 41 42 43 44 45 46 47 48 49
50 51 52
1 5 8 10 12 14 15 16 17 18 19 20 21 22 24 25 26 27 28 29 30 31 41 43 46 48
49 51 52 53 54 55 56 57 58 59 60 61
5 8 10 12 14 15 16 17 18 21 22 24 25 26 27 28 29 31 33 36 38 39 41 43 46 5
6 62 63 64 65 66 67 68
7 8 10 12 17 18 21 23 24 26 27 28 29 30 33 34 35 36 38 41 43 47 59 63 66 6
9 70 71 72 73 74 75 76 77 78 79
1 12 14 15 16 17 18 21 22 23 24 25 27 28 29 30 31 35 38 41 43 44 53 56 57
58 59 60 63 66 80 81 82 83 84
10 12 14 15 16 17 18 21 22 24 25 26 27 28 29 30 31 33 39 41 43 44 46 49 59
60 62 63 66 82
1 8 10 12 14 15 16 17 18 21 22 23 24 25 27 29 30 31 38 41 43 53 56 59 61 6
3 66 68 85 86 87 88 89
1 8 12 13 14 15 16 17 18 22 24 25 28 30 38 41 42 43 46 49 60 63 64 66 80 8
2 84 90 91 92 93 94 95
```

Question 1a: . How many items are there in the data?

In [2]:

```
!awk -- '{ for (i=1; i <= NF; i++) item[$i] += 1}; END {print length(item)}' accidents_
10k.dat
```

```
310
```

Answer: 310 items are there in the data.

Question 1b: How many transactions are present in the data?

In [3]:

```
!wc -l accidents_10k.dat
```

```
10000 accidents_10k.dat
```

Answer: 10000 transactions are there in the dataset 'accidents_10k.dat'.

Question 1c: . What is the length of the smallest transaction?

In [4]:

```
#!awk '{print NF}' accidents_10k.dat >length_of_transaction
#!sort -n length_of_transaction |uniq -c
#!awk '{print NF}' accidents_10k.dat|sort -n |head -1 -working
#!awk 'BEGIN {minlen=9999999} { if(minlen > NF) {minlen=NF}} END{print minlen}' acciden
ts_10k.dat --working
!awk '{print NR,NF}' accidents_10k.dat|sort -nk2 |head -1
```

```
7373 23
sort: write failed: standard output: Broken pipe
sort: write error
```

Answer: 23 is the length of the smallest transaction.

Question 1d: What is the length of the longest transaction?

In [5]:

```
#!awk '{print NF}' accidents_10k.dat >length_of_transaction
!awk 'BEGIN {max_len=0} { if(max_len < NF) {max_len=NF}} END{print max_len}' accidents_
10k.dat

#!sort -n length_of_transaction |tail
```

```
45
```

Answer: 45 is the length of the longest transaction.

Question 1e: What is the size of the search space of frequent itemsets in this data?

In [6]:

```
k=2**310
print('The size of the search space of frequent sets is 2^310 which is equal to ' +str(
k))
```

```
The size of the search space of frequent sets is 2^310 which is equal to 2
085924839766513752338888384931203236916703635113918720651407820138886450957
7656787131798913024
```

Answer: 2^310 is the size of the search space for frequent itemsets in the data.

Question 1f: Assume that you work for the deparment of transportation that collected this data. What benefit do you see in using itemset mining approaches on this data?

Answer: From the given data set we can get answers/solutions for some metrics which helps the department of transportation in imporoving the experirence /safety of the users.In total they were 572 attributes across various categories like environmental conditions,road conditions,human conditions,time of the accident etc.

Consider the attributes which fall under road conditions(dry road surface; wet road surface; snow on road surface;clean road surface; dirty road surface) , assigned 15,16,17,18,19 as the values for those attributes similarly Day of week (1 Monday; 2 Tuesday; 3 Wednesday; 4 Thursday; 5 Friday; 6 Saturday; 7 Sunday).By looking at the transactions in the accident dataset we can infer which attribute played a major role in the occurence of the accident .Example:if the attributes 17 ,6 , 7 repeated in most of the transactions we can infer snow on road surfaces is not cleared during the weekends and it causing the accidents and will appoint more workforce during weekends to avoid accidents.

Question 1g: What type of itemsets (frequent, maximial or closed) would you be interested in discovering this dataset? State your reason.

Answer: I am interested in discovering the closed frequent itemsets from the datasets because from closed frequent itemsets we can answer the questions: 1)Any random set item set which is subset of I is frequent or not and if frequent what is the support of that frequent itemset.

Maximal itemset(M) is a subset of closed itemset(C) is a subset of Frequent itemset.

Question 1h: What minsup threshold would you use and why?

Answer:

2. Generating frequent, maximal and closed itemsets using Apriori, ECLAT, and FPGrowth algorihtms from the dataset accidents_10k.dat

Question 2a: Generate frequent itemsets using Apriori, for minsup = 2000, 3000, and 4000. Which of these minsup thresholds results in a maximum number of frequent itemsets? Which of these minsup thresholds results in a least number of frequent itemsets? Provide a rationale for these observations.

In [7]:

```
!./apriori -tc -s-4000 accidents_10k.dat accident_frequent_4000.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
01s].
filtering, sorting and recoding items ... [33 item(s)] done [0.01s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.
00s].
building transaction tree ... [22267 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.21s].
filtering for closed item sets ... done [0.01s].
writing accident_frequent_4000.dat ... [26415 set(s)] done [0.01s].
```

In [8]:

```
!./apriori -tc -s-2000 accidents_10k.dat accident_frequent_2000.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
00s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 14 done [26.74s].
filtering for closed item sets ... done [0.42s].
writing accident_frequent_2000.dat ... [519902 set(s)] done [0.09s].
```

In [9]:

```
!./apriori -tc -s-3000 accidents_10k.dat accident_frequent_3000.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
01s].
filtering, sorting and recoding items ... [38 item(s)] done [0.00s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.
01s].
building transaction tree ... [24741 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 done [5.10s].
filtering for closed item sets ... done [0.05s].
writing accident_frequent_3000.dat ... [104269 set(s)] done [0.01s].
```

Answer: when minsup 4000 the number of frequent itemsets are 26415 which is less when compared with support values 3000 and 2000 this is because when support value increases we are eliminating the itemsets with in the support range in between 2000 and 3999.

Question 2b: Using Apriori, compare the execution time for finding frequent itemsets for minsup = 2000, 3000, and 4000. Which of these minsup thresholds takes the least amount of time? Provide a rationale for this observation.

In [10]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
01s].
building transaction tree ... [20250 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [18.27s].
writing <null> ... [851034 set(s)] done [0.02s].
(18, 'secs ', 926604, 'microsecs')
```

In [11]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-3000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
01s].
filtering, sorting and recoding items ... [38 item(s)] done [0.00s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.
01s].
building transaction tree ... [24741 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 done [4.14s].
writing <null> ... [133799 set(s)] done [0.00s].
(4, 'secs ', 494978, 'microsecs')
```

In [12]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.
01s].
building transaction tree ... [22267 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.19s].
writing <null> ... [29501 set(s)] done [0.00s].
(1, 'secs ', 485774, 'microsecs')
```

Answer: when minsupport value 4000 ,the execution time is only one sec and 681925 micro sec which is very less when compared with min support values 2000 and 3000 this is because the apriori used bfs approach and prunes all supersets of any infrequent candidate.Example when support equal to 4000 the level one sets i.e the item sets with length one also should have a support equals to 4000 for proceeding further to calculate the support of there supersets and as min support values increases from 2000 to 4000 the early level(L1,L2,L3,...LN) frequent item sets also decreases because we loose the itemsets with support 2000 to 3999 as it was not needed to calculate the support of the superstets of these infrequent itemsets which in turn reduces the computational time/executional time.

Question 2c: Using Apriori, find the frequent itemsets for minsup = 2000, 3000, and 4000. Determine the number of itemsets for each size (1 to max length of an itemset). What trends do you see that are common for all three minsup thresholds? What trends do you see that are different? Provide a rationale for these observations.

In [13]:

```
!awk '{print NF-1}' accident_frequent_2000.dat |sort -n |uniq -c
```

```
     29 1
    370 2
   2702 3
  12573 4
  38222 5
  79042 6
 114992 7
 119533 8
  88250 9
  45043 10
  15322 11
   3368 12
    435 13
     21 14
```

In [14]:

```
!awk '{print NF-1}' accident_frequent_3000.dat |sort -n |uniq -c
```

```
     27 1
    320 2
   2009 3
   7314 4
  16578 5
  24728 6
  25028 7
  17364 8
   8122 9
   2390 10
    369 11
     20 12
```

In [15]:

```
!awk '{print NF-1}' accident_frequent_4000.dat |sort -n |uniq -c
```

```
     27 1
    265 2
   1288 3
   3552 4
   6140 5
   6926 6
   5109 7
   2374 8
    640 9
     88 10
      6 11
```

Answer: From the above three outputs there is a drop in the number of number of frequent item sets for the corresponding lengths when min-support equals to 2000 we have item sets with length 13 and 14 and when min-support equal to 3000 we couldn't see items with length 13 and 14 similarly when min-support equals to 4000 there no frequent item sets with lenght 12 ,13,14 this is because as minsup value increases the itemsets with length 12,13,14 failed to meet the condition sup(itemsets of length(12,13,14))>=minimum support threshold(2000,3000,4000) and the trends that are common in differnce in

Question 2d: Using Apriori with minsup=2000, compare the number of frequent, maximal, and closed itemsets. Which is the largest set and which is the smallest set? Provide a rationale for these observations.

In [16]:

```
!./apriori -ts -s-2000 accidents_10k.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
01s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [17.77s].
writing <null> ... [851034 set(s)] done [0.01s].
```

In [17]:

```
!./apriori -tc -s-2000 accidents_10k.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
01s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 14 done [26.84s].
filtering for closed item sets ... done [0.43s].
writing <null> ... [519902 set(s)] done [0.01s].
```

In [18]:

```
!./apriori -tm -s-2000 accidents_10k.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
01s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
01s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 14 done [26.72s].
filtering for maximal item sets ... done [0.03s].
writing <null> ... [12330 set(s)] done [0.01s].
```

Answer: Frequent itemset(F) is the largest item set containing 851034 items and maximal item set(M) is the smallest item set with 12330 items this is because maximal itemset(M) is a subset of closed itemset(C) and closed item set is a subset of frequent itemset(F).

Question 2e: For a minsup = 2000, compare the execution time for Apriori, ECLAT and FPGrowth. Which of these algorithms took the least amount of time. Provide a rationale for this observation.

In [19]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

import datetime
start = datetime.datetime.now()
!./eclat -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

import datetime
start = datetime.datetime.now()
!./fpgrowth -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
03s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
00s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [17.93s].
writing <null> ... [851034 set(s)] done [0.01s].
(18, 'secs ', 532829, 'microsecs')
./eclat - find frequent item sets with the eclat algorithm
version 5.20 (2017.05.30)        (c) 2002-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
01s].
writing <null> ... [851034 set(s)] done [0.26s].
(0, 'secs ', 564339, 'microsecs')
./fpgrowth - find frequent item sets with the fpgrowth algorithm
version 6.17 (2017.05.30)        (c) 2004-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.
01s].
writing <null> ... [851034 set(s)] done [0.08s].
(0, 'secs ', 390183, 'microsecs')
```

Answer: FP growth algorithm took the least of execution time for generation of frequent item set with support value 2000 because the FP growth algorithm scans the DB till the construction the FP tree and after that all frequent item sets will be mined from the tree directly with out scanning the DB.so the took less execution when compared with other algorithms.

The reason for the elcat alogrithm to take more execution time when compared with FP tree because it needed one DB scan and once the Tid's were calculated for each item the next level item sets were obtained from the current level when minsupport =2000 the transaction set id's will be large and to derive next level item sets it take more time.

Question 2f: For a minsup = 4000, compare the execution time for Apriori, ECLAT and FPGrowth. Which of these algorithms took the least amount of time. Provide a rationale for this observation.

In [20]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

import datetime
start = datetime.datetime.now()
!./eclat -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

import datetime
start = datetime.datetime.now()
!./fpgrowth -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.
01s].
building transaction tree ... [22267 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.21s].
writing <null> ... [29501 set(s)] done [0.00s].
(1, 'secs ', 501080, 'microsecs')
./eclat - find frequent item sets with the eclat algorithm
version 5.20 (2017.05.30)        (c) 2002-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.
01s].
writing <null> ... [29501 set(s)] done [0.04s].
(0, 'secs ', 315245, 'microsecs')
./fpgrowth - find frequent item sets with the fpgrowth algorithm
version 6.17 (2017.05.30)        (c) 2004-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.
00s].
writing <null> ... [29501 set(s)] done [0.03s].
(0, 'secs ', 290086, 'microsecs')
```

Answer: FP Growth algorithm took the least amount of time when compared with apriori and elcat algorithm when support value is 4000 because in FP Growth no need to load DB after the construction of FP tree and it stores in the memory (no need to access secondary disks) and support for the itemsets computed using that tree ,as it requires less DB scans and no need of accessing the disk .

Question 2g: For a minsup = 6000, compare the execution time for Apriori, ECLAT and FPGrowth. Which of these algorithms took the least amount of time. Provide a rationale for this observation.

In [21]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-6000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

import datetime
start = datetime.datetime.now()
!./eclat -ts -s-6000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

import datetime
start = datetime.datetime.now()
!./fpgrowth -ts -s-6000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.
00s].
building transaction tree ... [6478 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 done [0.04s].
writing <null> ... [2254 set(s)] done [0.00s].
(0, 'secs ', 292724, 'microsecs')
./eclat - find frequent item sets with the eclat algorithm
version 5.20 (2017.05.30)        (c) 2002-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.
00s].
writing <null> ... [2254 set(s)] done [0.01s].
(0, 'secs ', 279384, 'microsecs')
./fpgrowth - find frequent item sets with the fpgrowth algorithm
version 6.17 (2017.05.30)        (c) 2004-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.
02s].
filtering, sorting and recoding items ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.
00s].
writing <null> ... [2254 set(s)] done [0.00s].
(0, 'secs ', 266689, 'microsecs')
```

Answer: Eclat algorithm uses the DFS search startegy tooks the least of time when compared with apriori and fp growth because when support is 6000 the transaction id sets will be small(when compared with 2000 and 4000) as the tids will be small it takes less time to compute the next level item frequent sets.

Question 2h: Fill the following table based on execution times computed in **2e**, **2f**, and **2g**. State your observations on the relative computational efficiency at different support thresholds. Based on your knowledge of these algorithms, provide the reasons behind your observations.

Algorithm	**minsup=2000**	**minsup=4000**	**minsup=6000**
Apriori	20 secs and 180335 microsecs	one sec and 602734 microssecs	301802 microssecs
Eclat	554653 microsecs	309742 microsecs	272680 microssecs
FPGrowth	366921 microsecs	306384 microsecs	293593 microsecs

Answer: Apriori algorithm is an iterative approach known as level-wise approach and uses the BFS startegy and it is optimal interms of the candiates that we count ,repteadely scans the database at every level for calculating the support of the candiate sets and using of apriori would be good when the database size is huge and need frequent sets at high (minimum support).Here for Apriori the execution time decreases as min support value incereases from 2000 to 6000.

Eclat algorithm uses DFS startegy and it uses the vertical database layout and less space than apriori if item sets are less in number ,requires one DB scan for the calculation of tids sets and next level will be calculated using the current level Tid's and it would used when database is small and needs frequent sets with high (minimum support). when minsup =6000 it took less time because the tid's will be small as we eliminate the postings from 2000 to 5999 and as DB is small

3. Discovering frequent subsequences and substrings

Assume that roads in a Cincinnati are assigned numbers. Participants are enrolled in a transportation study and for every trip they make using their car, the sequence of roads taken are recorded. Trips that involves freeways are excluded. This data is in the file road_seq_data.dat.

Question 3a: What 'type' of sequence mining will you perform to determine frequently taken 'paths'? Paths are sequences of roads traveresed consecutively in the same order.

Answer: substring mining would be useful to find the roads traveresed consecutively (with out any gaps).

Question 3b: How many sequences are there in this sequence database?

In [22]:

```
!wc -l road_seq_data.dat
```

```
1000 road_seq_data.dat
```

In [23]:

```
!head road_seq_data.dat
```

```
74731 74748 74783 109554 1
74731 74747 105488 13 30 65 34836 37601
74731 74736 92326 13 29 30770 37613 37630 37665 71994
74731 74756 74957 13 18 17608 37613 37629 67992
74731 74760 74782 95201 13 38 239 37613 37618 55002
74725 74774 78147 13 42 64 20483 37613 37638 37839
74731 74736 111385 7 56 3429 37613 37642 37664 57829
74723 74744 74867 13 18 36667 37607 37656 40995
74731 74748 74782 74791 110597 5 26 149 37613 37618 73797
74731 74748 74782 74792 111926 13 30 64 73 35879 37605 37626 37749
```

Answer: 1000 sequences are there in the database.

Question 3c: What is the size of the alphabet in this sequence database?

In [24]:

```
#!cat road_seq_data.dat|xargs|tr -s ' ' '\n'|sort|uniq|tr -s  '\n' ',' >alphabet
#!cat road_seq_data.dat|xargs|tr -s ' ' '\n'|sort|uniq -c|awk '{if($2==100){print $0}}'
!cat road_seq_data.dat|xargs|tr -s ' ' '\n'|sort|uniq -c|wc -l
#!cat road_seq_data.dat|xargs|tr -s ' ' '\n'|sort|uniq -c
```

```
1283
```

Answer: 1283 is the size of the alphabet in this sequence database.

Question 3d: What are the total number of possible subsequences of length 2 in this dataset?

In [25]:

```
print('Total possible number of subsequences of length 2 in the dataset ' + str(1283*12
83) +' because we can travel across any road out of 1283 roads ,we can choose the first
road in 1283 ways and also second road in 1283 ways ')
```

```
Total possible number of subsequences of length 2 in the dataset 1646089 b
ecause we can travel across any road out of 1283 roads ,we can choose the
first road in 1283 ways and also second road in 1283 ways
```

Question 3e: What are the total number of possible substrings of length 2 in this dataset?

In [26]:

```
#!awk -- 'BEGIN {substring=0  } { for (i=0 ; i <= 1000 ; i++) {substring=(NF-1)+substri
ng}  }; END {print substring}' road_seq_data.dat-- not working
!awk  '{sum+=(NF-1);} END {print sum}' road_seq_data.dat
```

```
8940
```

8940 number of possible substrings are available in the dataset of length 2. here the window size is 2 and for the first sequence S1=NF+2-1 where 2 is the window size so the result becomes NF-1 and similarly for next sequence also NF-1 and if we add all the sequences (number of fields -1) value ,the result would be the possible number of substrings of length 2.

Question 3f: Discover frequent **subsequences** with minsup = 10 and report the number of subsequences discovered.

In [27]:

```
!./prefixspan -min_sup 10 road_seq_data.dat |sed -n 'p;n' |wc  -l
```

```
PrefixSpan version 1.00 - Sequential Pattern Miner
Written by Yasuo Tabei

4589
```

Answer: 4589 are the frequent subsequences discovered when mimimum support =10.

Question 3g: Discover frequent **substrings** with minsup = 10 and report the number of substrings discovered.

In [28]:

```
!./seqwog -ts -s-10 road_seq_data.dat substring_assign_results
```

```
./seqwog - find frequent sequences without gaps
version 3.16 (2016.10.15)        (c) 2010-2016   Christian Borgelt
reading road_seq_data.dat ... [1283 item(s), 1000 transaction(s)] done [0.
00s].
recoding items ... [1283 item(s)] done [0.00s].
reducing and triming transactions ... [844/1000 transaction(s)] done [0.00
s].
writing substring_assign_results ... [613 sequence(s)] done [0.01s].
```

In [29]:

```
!cat substring_assign_results
```

```
95201 13 (10)
95201 (10)
77751 (10)
20483 37613 (10)
20483 (10)
3033 (10)
74826 74848 77336 11 (10)
74826 74848 77336 (10)
74826 74848 (10)
74826 (10)
77336 11 (10)
77336 (10)
40602 (10)
40191 (10)
37708 37730 40191 (10)
37708 37730 (10)
37708 (10)
108 130 2618 37611 (10)
108 130 2618 (10)
108 130 (10)
108 (10)
2618 37611 (10)
2618 (10)
37679 (11)
74797 (11)
77184 13 16 (11)
77184 13 (11)
77184 (11)
40560 (11)
79 (11)
2466 37613 37616 (11)
2466 37613 (11)
2466 (11)
2991 37611 37631 (10)
2991 37611 (11)
2991 (11)
77709 11 31 (10)
77709 11 (11)
77709 (11)
40039 (11)
74767 (13)
37641 40560 (11)
37641 (13)
74784 (13)
49 (13)
39298 (13)
74759 77709 11 31 (10)
74759 77709 11 (11)
74759 77709 (11)
74759 (13)
37649 (13)
37666 (13)
41 2991 37611 37631 (10)
41 2991 37611 (11)
41 2991 (11)
41 (13)
66 (13)
37674 (14)
76435 13 (12)
76435 (14)
1717 37613 (12)
```

```
1717 (14)
74792 (14)
74 (14)
74949 13 (12)
74949 (14)
37831 (14)
231 37613 (12)
231 (14)
40579 (15)
3010 37613 (13)
3010 (15)
77728 13 (13)
77728 (15)
74891 13 30 (11)
74891 13 (16)
74891 (16)
74888 13 30 64 73 (10)
74888 13 30 64 (10)
74888 13 30 (10)
74888 13 (16)
74888 (16)
173 37613 37630 (11)
173 37613 (16)
173 (16)
170 37613 37630 37664 37673 (10)
170 37613 37630 37664 (10)
170 37613 37630 (10)
170 37613 (16)
170 (16)
37773 (16)
37770 (16)
74867 1 (13)
74867 (17)
74744 74867 1 (13)
74744 74867 (17)
74744 (17)
26 149 37601 (13)
26 149 (17)
26 (17)
149 37601 (13)
149 (17)
37626 37749 (17)
37626 (17)
37749 (17)
74726 74826 74848 77336 11 (10)
74726 74826 74848 77336 (10)
74726 74826 74848 (10)
74726 74826 (10)
74726 (20)
8 108 130 2618 37611 (10)
8 108 130 2618 (10)
8 108 130 (10)
8 108 (10)
8 (20)
37608 37708 37730 40191 (10)
37608 37708 37730 (10)
37608 37708 (10)
37608 (20)
74896 (20)
178 (20)
37778 (20)
```

```
74897 13 18 (10)
74897 13 (16)
74897 (20)
179 37613 37618 (10)
179 37613 (16)
179 (20)
37779 (20)
79279 13 (18)
79279 (21)
4561 37613 (18)
4561 (21)
41039 (21)
3473 37613 (17)
3473 (21)
42109 (21)
78191 13 (17)
78191 (21)
3320 37613 37618 (13)
3320 37613 (21)
3320 (21)
40886 (21)
78038 13 18 (13)
78038 13 (21)
78038 (21)
68685 (22)
31471 37613 37638 (21)
31471 37613 (21)
31471 (22)
106189 13 38 (21)
106189 13 (21)
106189 (22)
74745 78191 13 (17)
74745 78191 (21)
74745 (24)
27 3473 37613 (17)
27 3473 (21)
27 (24)
37627 41039 (21)
37627 (24)
37681 37748 (17)
37681 (26)
81 148 (17)
81 (26)
37748 (27)
148 (27)
74866 (27)
74799 74866 (17)
74799 (27)
74791 (30)
73 (30)
37673 (30)
37730 40191 (10)
37730 (31)
130 2618 37611 (10)
130 2618 (10)
130 (31)
74848 77336 11 (10)
74848 77336 (10)
74848 (32)
37665 (36)
37656 40995 (36)
```

```
37656 (36)
74774 78147 13 18 (12)
74774 78147 13 42 (10)
74774 78147 13 (32)
74774 78147 (36)
74774 (36)
74783 (36)
56 3429 37613 37618 (12)
56 3429 37613 37642 (10)
56 3429 37613 (32)
56 3429 (36)
56 (36)
65 (36)
40995 (37)
78147 13 18 (12)
78147 13 42 (10)
78147 13 (32)
78147 (37)
3429 37613 37618 (12)
3429 37613 37642 (10)
3429 37613 (32)
3429 (37)
150 37613 (17)
150 (38)
22 179 37613 37618 (10)
22 179 37613 (16)
22 179 (20)
22 173 37613 37630 (11)
22 173 37613 (16)
22 173 (16)
22 (38)
37622 37779 (20)
37622 37773 (16)
37622 (38)
37648 37750 (38)
37648 (38)
37750 (38)
74766 74868 13 (17)
74766 74868 (38)
74766 (38)
48 150 37613 (17)
48 150 (38)
48 (38)
74868 13 (17)
74868 (38)
74739 74896 (20)
74739 74888 13 30 64 73 (10)
74739 74888 13 30 64 (10)
74739 74888 13 30 (10)
74739 74888 13 (16)
74739 74888 (16)
74739 (38)
21 178 (20)
21 170 37613 37630 37664 37673 (10)
21 170 37613 37630 37664 (10)
21 170 37613 37630 (10)
21 170 37613 (16)
21 170 (16)
21 (38)
37621 37778 (20)
37621 37770 (16)
```

```
37621 (38)
74740 74897 13 18 (10)
74740 74897 13 (16)
74740 74897 (20)
74740 74891 13 30 (11)
74740 74891 13 (16)
74740 74891 (16)
74740 (38)
31 81 148 (12)
31 81 (14)
31 79 (11)
31 (43)
37631 37681 37748 (12)
37631 37681 (14)
37631 37679 (11)
37631 (43)
74749 74799 74866 (12)
74749 74799 (15)
74749 74797 (11)
74749 (44)
151 37613 37618 (10)
151 37613 (30)
151 (46)
25 151 37613 37618 (10)
25 151 37613 (30)
25 151 (46)
25 (46)
74869 13 18 (10)
74869 13 (30)
74869 (46)
74743 74869 13 18 (10)
74743 74869 13 (30)
74743 74869 (46)
74743 (46)
62 31471 37613 37638 (21)
62 31471 37613 (21)
62 31471 (21)
62 (47)
37662 68685 (21)
37662 (47)
74780 106189 13 38 (21)
74780 106189 13 (21)
74780 106189 (21)
74780 (47)
74719 5 25 151 37613 37618 (10)
74719 5 25 151 37613 (30)
74719 5 25 151 (46)
74719 5 25 (46)
74719 5 (46)
74719 (49)
37601 (50)
1 (50)
37664 37673 (30)
37664 37674 (14)
37664 (53)
74782 74791 (30)
74782 74792 (14)
74782 95201 13 (10)
74782 95201 (10)
74782 (54)
64 73 (30)
```

```
64 74 (14)
64 20483 37613 (10)
64 20483 (10)
64 (54)
11 31 81 148 (12)
11 31 81 (14)
11 31 79 (11)
11 31 (42)
11 (56)
37611 37631 37681 37748 (12)
37611 37631 37681 (14)
37611 37631 37679 (11)
37611 37631 (42)
37611 (56)
74729 74749 74799 74866 (12)
74729 74749 74799 (15)
74729 74749 74797 (11)
74729 74749 (43)
74729 (57)
37616 (59)
74734 (60)
16 (60)
37620 37658 40886 (21)
37620 37658 42109 (21)
37620 37658 40579 (15)
37620 37658 39298 (12)
37620 37658 (72)
37620 (79)
74738 74776 78038 13 18 (13)
74738 74776 78038 13 (21)
74738 74776 78038 (21)
74738 74776 79279 13 (18)
74738 74776 79279 (21)
74738 74776 77728 13 (13)
74738 74776 77728 (15)
74738 74776 76435 13 (12)
74738 74776 76435 (13)
74738 74776 (74)
74738 (81)
20 58 3320 37613 37618 (13)
20 58 3320 37613 (21)
20 58 3320 (21)
20 58 4561 37613 (18)
20 58 4561 (21)
20 58 3010 37613 (13)
20 58 3010 (15)
20 58 1717 37613 (12)
20 58 1717 (13)
20 58 (74)
20 (81)
38 231 37613 (12)
38 231 (14)
38 (89)
37638 37831 (14)
37638 (89)
74756 74949 13 (12)
74756 74949 (14)
74756 (89)
37642 37658 (28)
37642 37662 68685 (21)
37642 37662 (47)
```

```
37642 (97)
74760 74776 (28)
74760 74782 95201 13 (10)
74760 74782 95201 (10)
74760 74782 (10)
74760 74780 106189 13 38 (21)
74760 74780 106189 13 (21)
74760 74780 106189 (21)
74760 74780 (47)
74760 (98)
42 58 (28)
42 64 20483 37613 (10)
42 64 20483 (10)
42 64 (10)
42 62 31471 37613 37638 (21)
42 62 31471 37613 (21)
42 62 31471 (21)
42 62 (47)
42 (98)
37630 37658 (11)
37630 37664 37673 (30)
37630 37664 37674 (14)
37630 37664 (44)
37630 37665 (36)
37630 (100)
74748 74776 (11)
74748 74782 74791 (30)
74748 74782 74792 (14)
74748 74782 (44)
74748 74783 (36)
74748 (100)
30 58 (11)
30 64 73 (30)
30 64 74 (14)
30 64 (44)
30 65 (36)
30 (100)
37658 40886 (21)
37658 42109 (21)
37658 40579 (15)
37658 39298 (12)
37658 (114)
58 3320 37613 37618 (13)
58 3320 37613 (21)
58 3320 (21)
58 4561 37613 (18)
58 4561 (21)
58 3010 37613 (13)
58 3010 (15)
58 1717 37613 (12)
58 1717 (13)
58 (116)
74776 78038 13 18 (13)
74776 78038 13 (21)
74776 78038 (21)
74776 79279 13 (18)
74776 79279 (21)
74776 77728 13 (13)
74776 77728 (15)
74776 76435 13 (12)
74776 76435 (13)
```

```
74776 (116)
37605 37618 (10)
37605 37621 37778 (20)
37605 37621 37770 (16)
37605 37621 (38)
37605 37648 37750 (38)
37605 37648 (38)
37605 37622 37779 (20)
37605 37622 37773 (16)
37605 37622 (38)
37605 37626 37749 (17)
37605 37626 (17)
37605 (141)
18 (151)
37618 (151)
74736 (151)
37607 37620 37658 40886 (21)
37607 37620 37658 42109 (21)
37607 37620 37658 40579 (15)
37607 37620 37658 39298 (12)
37607 37620 37658 (72)
37607 37620 (79)
37607 37656 40995 (36)
37607 37656 (36)
37607 37627 41039 (21)
37607 37627 (24)
37607 37641 40560 (11)
37607 37641 (13)
37607 (168)
7 20 58 3320 37613 37618 (13)
7 20 58 3320 37613 (21)
7 20 58 3320 (21)
7 20 58 4561 37613 (18)
7 20 58 4561 (21)
7 20 58 3010 37613 (13)
7 20 58 3010 (15)
7 20 58 1717 37613 (12)
7 20 58 1717 (13)
7 20 58 (74)
7 20 (81)
7 56 3429 37613 37618 (12)
7 56 3429 37613 37642 (10)
7 56 3429 37613 (32)
7 56 3429 (36)
7 56 (36)
7 27 3473 37613 (17)
7 27 3473 (21)
7 27 (24)
7 41 2991 37611 37631 (10)
7 41 2991 37611 (11)
7 41 2991 (11)
7 41 (13)
7 (170)
74725 74738 74776 78038 13 18 (13)
74725 74738 74776 78038 13 (21)
74725 74738 74776 78038 (21)
74725 74738 74776 79279 13 (18)
74725 74738 74776 79279 (21)
74725 74738 74776 77728 13 (13)
74725 74738 74776 77728 (15)
74725 74738 74776 76435 13 (12)
```

```
74725 74738 74776 76435 (13)
74725 74738 74776 (74)
74725 74738 (81)
74725 74774 78147 13 18 (12)
74725 74774 78147 13 42 (10)
74725 74774 78147 13 (32)
74725 74774 78147 (36)
74725 74774 (36)
74725 74745 78191 13 (17)
74725 74745 78191 (21)
74725 74745 (24)
74725 74759 77709 11 31 (10)
74725 74759 77709 11 (11)
74725 74759 77709 (11)
74725 74759 (13)
74725 (170)
74723 74736 (10)
74723 74743 74869 13 18 (10)
74723 74743 74869 13 (30)
74723 74743 74869 (46)
74723 74743 (46)
74723 74740 74897 13 18 (10)
74723 74740 74897 13 (16)
74723 74740 74897 (20)
74723 74740 74891 13 30 (11)
74723 74740 74891 13 (16)
74723 74740 74891 (16)
74723 74740 (38)
74723 74739 74896 (20)
74723 74739 74888 13 30 64 73 (10)
74723 74739 74888 13 30 64 (10)
74723 74739 74888 13 30 (10)
74723 74739 74888 13 (16)
74723 74739 74888 (16)
74723 74739 (38)
74723 74766 74868 13 (17)
74723 74766 74868 (38)
74723 74766 (38)
74723 74744 74867 1 (13)
74723 74744 74867 (17)
74723 74744 (17)
74723 (187)
5 18 (10)
5 25 151 37613 37618 (10)
5 25 151 37613 (30)
5 25 151 (46)
5 25 (46)
5 21 178 (20)
5 21 170 37613 37630 37664 37673 (10)
5 21 170 37613 37630 37664 (10)
5 21 170 37613 37630 (10)
5 21 170 37613 (16)
5 21 170 (16)
5 21 (38)
5 48 150 37613 (17)
5 48 150 (38)
5 48 (38)
5 22 179 37613 37618 (10)
5 22 179 37613 (16)
5 22 179 (20)
5 22 173 37613 37630 (11)
```

```
5 22 173 37613 (16)
5 22 173 (16)
5 22 (38)
5 26 149 37601 (13)
5 26 149 (17)
5 26 (17)
5 (187)
37613 37618 (138)
37613 37630 37658 (11)
37613 37630 37664 37673 (30)
37613 37630 37664 37674 (14)
37613 37630 37664 (44)
37613 37630 37665 (36)
37613 37630 (100)
37613 37642 37658 (28)
37613 37642 37662 68685 (21)
37613 37642 37662 (47)
37613 37642 (90)
37613 37638 37831 (14)
37613 37638 (89)
37613 37616 (59)
37613 (499)
13 18 (138)
13 30 58 (11)
13 30 64 73 (30)
13 30 64 74 (14)
13 30 64 (44)
13 30 65 (36)
13 30 (100)
13 42 58 (28)
13 42 64 20483 37613 (10)
13 42 64 20483 (10)
13 42 64 (10)
13 42 62 31471 37613 37638 (21)
13 42 62 31471 37613 (21)
13 42 62 31471 (21)
13 42 62 (47)
13 42 (91)
13 38 231 37613 (12)
13 38 231 (14)
13 38 (89)
13 16 (60)
13 (501)
74731 74736 (138)
74731 74748 74776 (11)
74731 74748 74782 74791 (30)
74731 74748 74782 74792 (14)
74731 74748 74782 (44)
74731 74748 74783 (36)
74731 74748 (100)
74731 74760 74776 (28)
74731 74760 74782 95201 13 (10)
74731 74760 74782 95201 (10)
74731 74760 74782 (10)
74731 74760 74780 106189 13 38 (21)
74731 74760 74780 106189 13 (21)
74731 74760 74780 106189 (21)
74731 74760 74780 (47)
74731 74760 (91)
74731 74756 74949 13 (12)
74731 74756 74949 (14)
```

```
74731 74756 (89)
74731 74734 (60)
74731 (501)
```

Answer: 613 substrings were present in the given dataset with minimum support value 10.

Question 3h: Explain the difference in the number of frequent subsequences and substrings found in **3f** and **3g** above.

Answer: Frequent subsequences are more when compared with substrings for a minimum support value 10 because in the subsequences the symbols can come with gaps i.e. in our dataset for sequence s1:"74731 74748 74783 109554 1" we can choose the road after 74731 (1 or 74783 or 109554 or 74748 or 74731) to a form a subsequence of length 2 where as for substrings we can't choose a symbol other than it's consecutive symbol ,gaps are not allowed i.e for 74731 74748 only be the substring and from the sequence s1: we will be having 25 subsequences and 5-2+1=4 substrings ,search space is less for substrings when compared with subsequences .