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 3
0 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 37 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 5
1 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 56 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 69 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 6
2 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 63 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 82 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++) wc[$i] += 1}; END {print length(wc)}' acciden
```

```
310
```

Answer:310

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

In [16]:
```
!wc -l accidents_10k.dat
```

```
10000 accidents_10k.dat
```

```
<span style="color:green">**Answer:10000** </span>
```

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

In [37]:
```
!awk '{print NF}' accidents_10k.dat | sort
```

```
23
23
23
23
24
24
24
24
24
24
25
25
25
25
25
25
26
26
26
26
26
```

Answer:23

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

In [38]:
```
!awk '{print NF}' accidents_10k.dat | sort -r
```

```
45
45
45
45
45
44
44
44
44
44
44
44
43
43
43
43
43
43
43
43
```

Answer:45

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

In [20]:
```
!awk -- '{for (i = 1; i <= NF; i++) wc[$i] += 1}; END {print length(wc)}' acciden
```

```
310
```

Answer:2 ^ 310

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: We can use this data to extract many meaningful patterns such as given what condition , most accidents occur. we can also find which roads have the most deadly accidents and we can use information of this type to fix the road.

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

Answer: I'd be interested in closed itemsets as it would give me all information unlike maximal which would only tell me if a itemset is frequent or not and it'd also the save me the heavy computations required incase of frequent itemsets

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

Answer: I would select minsup between 166 - 200, because it's optimum. It's not too big to ignore important patterns and also not too small to needlessly include trivial patterns

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

In [7]:
```
!chmod u+x apriori
```

In [8]:
```
!./apriori
```

```
usage: ./apriori [options] infile [outfile]
find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
-t#      target type                              (default: s)
         (s: frequent, c: closed, m: maximal item sets,
          g: generators, r: association rules)
-m#      minimum number of items per set/rule     (default: 1)
-n#      maximum number of items per set/rule     (default: no limit)
-s#      minimum support of an item set/rule      (default: 10%)
-S#      maximum support of an item set/rule      (default: 100%)
         (positive: percentage, negative: absolute number)
-o       use original rule support definition     (body & head)
-c#      minimum confidence of an assoc. rule     (default: 80%)
-e#      additional evaluation measure            (default: none)
-a#      aggregation mode for evaluation measure  (default: none)
-d#      threshold for add. evaluation measure    (default: 10%)
-i       invalidate eval. below expected support  (default: evaluate all)
-p#      (min. size for) pruning with evaluation  (default: no pruning)
         (< 0: weak forward, > 0 strong forward, = 0: backward pruning)
-q#      sort items w.r.t. their frequency        (default: 2)
         (1: ascending, -1: descending, 0: do not sort,
          2: ascending, -2: descending w.r.t. transaction size sum)
-u#      filter unused items from transactions    (default: 0.01)
         (0: do not filter items w.r.t. usage in sets,
         <0: fraction of removed items for filtering,
         >0: take execution times ratio into account)
-x       do not prune with perfect extensions     (default: prune)
-y       a-posteriori pruning of infrequent item sets
-T       do not organize transactions as a prefix tree
-F#:#..  support border for filtering item sets   (default: none)
         (list of minimum support values, one per item set size,
         starting at the minimum size, as given with option -m#)
-R#      read item selection/appearance indicators
-P#      write a pattern spectrum to a file
-Z       print item set statistics (number of item sets per size)
-N       do not pre-format some integer numbers   (default: do)
-g       write item names in scanable form (quote certain characters)
-h#      record header  for output               (default: "")
-k#      item separator for output               (default: " ")
-I#      implication sign for association rules   (default: " <- ")
-v#      output format for set/rule information   (default: " (%S)")
-j#      sort item sets in output by their size   (default: no sorting)
         (< 0: descending, > 0: ascending order)
-w       integer transaction weight in last field (default: only items)
-r#      record/transaction separators            (default: "\n")
-f#      field /item       separators             (default: " \t,")
-b#      blank   characters                       (default: " \t\r")
-C#      comment characters                       (default: "#")
-!       print additional option information
infile   file to read transactions from           [required]
outfile  file to write item sets/assoc. rules to  [optional]
```

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 [9]:
```
!./apriori -ts -s-2000 accidents_10k.dat T_ACCIDENT_Freq_S2000.txt
```

```
./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 [19.38s].
writing T_ACCIDENT_Freq_S2000.txt ... [851034 set(s)] done [0.10s].
```

In [10]:
```
!wc -l T_ACCIDENT_Freq_S2000.txt
```

```
851034 T_ACCIDENT_Freq_S2000.txt
```

In [11]:
```
!./apriori -ts -s-3000 accidents_10k.dat T_ACCIDENT_Freq_S3000.txt
```

```
./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 [4.41s].
writing T_ACCIDENT_Freq_S3000.txt ... [133799 set(s)] done [0.02s].
```

In [12]:
```
!wc -l T_ACCIDENT_Freq_S3000.txt
```

```
133799 T_ACCIDENT_Freq_S3000.txt
```

In [13]:
```
!./apriori -ts -s-4000 accidents_10k.dat T_ACCIDENT_Freq_S4000.txt
```

```
./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 [1.26s].
writing T_ACCIDENT_Freq_S4000.txt ... [29501 set(s)] done [0.01s].
```

In [14]:
```
!wc -l T_ACCIDENT_Freq_S4000.txt
```

```
29501 T_ACCIDENT_Freq_S4000.txt
```

Answer: minsup 4000 < Minsup 3000 < minsup 2000

Rationale: As the minsup increases the number of itemsets satisying the minsup goes down, hence the massive reduction in number of frequent itemsets when minsup increases

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 [15]:
```
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 [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [19.55s].
writing <null> ... [851034 set(s)] done [0.02s].
20 secs  149878 microsecs
```

In [16]:
```
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 [0.02s].
filtering, sorting and recoding items ... [38 item(s)] done [0.00s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.00s].
building transaction tree ... [24741 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 done [4.43s].
writing <null> ... [133799 set(s)] done [0.00s].
4 secs  775529 microsecs
```

In [17]:
```
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 [0.01s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.00s].
building transaction tree ... [22267 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.33s].
writing <null> ... [29501 set(s)] done [0.00s].
1 secs  646342 microsecs
```

Answer: minsup 4000 < Minsup 3000 < minsup 2000

Rationale: This is happening becasue as the minsup increases the number of frequent itemset satisfying the minsup decreases dastardly leading to effective pruning of space, and thus leading to decrease in time taken

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 [18]:
```
!awk '{print NF-1}' T_ACCIDENT_Freq_S2000.txt|sort -n|uniq -c
```

```
     49 1
    705 2
   5285 3
  23745 4
  69647 5
 139628 6
 195730 7
 193299 8
 133819 9
  63937 10
  20497 11
   4189 12
    483 13
     21 14
```

In [19]:
```
!awk '{print NF-1}' T_ACCIDENT_Freq_S3000.txt|sort -n|uniq -c
```

```
     38 1
    468 2
   2830 3
   9887 4
  21779 5
  31964 6
  32020 7
  21862 8
   9839 9
   2705 10
    387 11
     20 12
```

In [20]:
```
!awk '{print NF-1}' T_ACCIDENT_Freq_S4000.txt|sort -n|uniq -c
```

```
     33 1
    319 2
   1492 3
   4043 4
   6926 5
   7751 6
   5626 7
   2546 8
    668 9
     91 10
      6 11
```

Answer: Similarity is that for each minsup the value gradually increases from 1 to 6 and then falls down afterwards. Difference is the drop in the number of frequent itemsets around the midpoint when support increases which can be explained by the increase in minsup

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 [21]:
```
!./apriori -tm -s-2000 accidents_10k.dat T_Accidents_Maximal_S2000.txt
```

```
./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 [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 14 done [30.87s].
filtering for maximal item sets ... done [0.03s].
writing T_Accidents_Maximal_S2000.txt ... [12330 set(s)] done [0.02s].
```

In [22]:
```
!wc -l T_Accidents_Maximal_S2000.txt
```

```
12330 T_Accidents_Maximal_S2000.txt
```

In [23]:
```
!./apriori -tc -s-2000 accidents_10k.dat T_Accidents_Close_S2000.txt
```

```
./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 [31.37s].
filtering for closed item sets ... done [0.48s].
writing T_Accidents_Close_S2000.txt ... [519902 set(s)] done [0.10s].
```

In [24]:
```
!wc -l T_Accidents_Closed_S2000.txt
```

```
519902 T_Accidents_Closed_S2000.txt
```

In [25]:
```
!wc -l T_ACCIDENT_Freq_S2000.txt
```

```
851034 T_ACCIDENT_Freq_S2000.txt
```

Answer: Maximal < Closed < Frequent.

Rationale: It's because Maximal itemsets are itemsets that have no frequent superset (i.e it's maximum with 2000 minsup), hence we've less number of itemsets here compared to Closed Itemsets which have itemsets that has no frequent superset with same support which again is very less compared to the usual frequent itemsets.

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 [26]:
```
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 [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [19.90s].
writing <null> ... [851034 set(s)] done [0.01s].
20 secs  495271 microsecs
```

In [27]:
```
!chmod u+x eclat
```

In [28]:
```
!./eclat
```

```
usage: ./eclat [options] infile [outfile]
find frequent item sets with the eclat algorithm
version 5.20 (2017.05.30)        (c) 2002-2017   Christian Borgelt
-t#      target type                              (default: s)
         (s: frequent, c: closed, m: maximal item sets,
          g: generators, r: association rules)
-m#      minimum number of items per set/rule     (default: 1)
-n#      maximum number of items per set/rule     (default: no limit)
-s#      minimum support of an item set/rule      (default: 10%)
-S#      maximum support of an item set/rule      (default: 100%)
         (positive: percentage, negative: absolute number)
-o       use original rule support definition     (body & head)
-c#      minimum confidence of an assoc. rule     (default: 80%)
-e#      additional evaluation measure            (default: none)
-a#      aggregation mode for evaluation measure  (default: none)
-d#      threshold for add. evaluation measure    (default: 10%)
-i       invalidate eval. below expected support  (default: evaluate all)
-p#      (min. size for) pruning with evaluation  (default: no pruning)
         (< 0: weak forward, > 0 strong forward, = 0: backward pruning)
-q#      sort items w.r.t. their frequency        (default: 2)
         (1: ascending, -1: descending, 0: do not sort,
          2: ascending, -2: descending w.r.t. transaction size sum)
-A#      variant of the eclat algorithm to use    (default: 'a')
-x       do not prune with perfect extensions     (default: prune)
-l#      number of items for k-items machine      (default: 16)
         (only for algorithm variants i,r,o,   options -Ai/-Ar/-Ao)
-j       do not sort items w.r.t. cond. support   (default: sort)
         (only for algorithm variants i,b,t,d, options -Ai/-Ab/-At/-Ad)
-y#      check extensions for closed/maximal sets (default: repository)
         (0: horizontal, > 0: vertical representation)
         (only with improved tid lists variant, option -Ai)
-u       do not use head union tail (hut) pruning (default: use hut)
         (only for maximal item sets, option -tm, not with option -Ab)
-F#:#..  support border for filtering item sets   (default: none)
         (list of minimum support values, one per item set size,
         starting at the minimum size, as given with option -m#)
-R#      read item selection/appearance indicators
-P#      write a pattern spectrum to a file
-Z       print item set statistics (number of item sets per size)
-N       do not pre-format some integer numbers   (default: do)
-g       write output in scanable form (quote certain characters)
-h#      record header  for output               (default: "")
-k#      item separator for output               (default: " ")
-I#      implication sign for association rules   (default: " <- ")
-v#      output format for item set information   (default: " (%S)")
-w       transaction weight in last field         (default: only items)
-r#      record/transaction separators           (default: "\n")
-f#      field /item        separators            (default: " \t,")
-b#      blank   characters                       (default: " \t\r")
-C#      comment characters                       (default: "#")
-T#      file to write transaction identifiers to (default: none)
-!       print additional option information
infile   file to read transactions from           [required]
outfile  file to write item sets/assoc.rules to   [optional]
```

In [29]:
```
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");
```

```
./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 [0.28s].
0 secs  572650 microsecs
```

In [30]:
```
!chmod u+x fpgrowth
```

In [31]:
```
!./fpgrowth
```

```
usage: ./fpgrowth [options] infile [outfile]
find frequent item sets with the fpgrowth algorithm
version 6.17 (2017.05.30)        (c) 2004-2017   Christian Borgelt
-t#      target type                              (default: s)
         (s: frequent, c: closed, m: maximal item sets,
          g: generators, r: association rules)
-m#      minimum number of items per set/rule     (default: 1)
-n#      maximum number of items per set/rule     (default: no limit)
-s#      minimum support of an item set/rule      (default: 10%)
-S#      maximum support of an item set/rule      (default: 100%)
         (positive: percentage, negative: absolute number)
-o       use original rule support definition     (body & head)
-c#      minimum confidence of an assoc. rule     (default: 80%)
-e#      additional evaluation measure            (default: none)
-a#      aggregation mode for evaluation measure  (default: none)
-d#      threshold for add. evaluation measure    (default: 10%)
-i       invalidate eval. below expected support  (default: evaluate all)
-p#      (min. size for) pruning with evaluation  (default: no pruning)
         (< 0: weak forward, > 0 strong forward, = 0: backward pruning)
-q#      sort items w.r.t. their frequency        (default: 2)
         (1: ascending, -1: descending, 0: do not sort,
          2: ascending, -2: descending w.r.t. transaction size sum)
-A#      variant of the fpgrowth algorithm to use (default: c)
-x       do not prune with perfect extensions     (default: prune)
-l#      number of items for k-items machine      (default: 16)
         (only for variants s and d, options -As or -Ad)
-j       do not sort items w.r.t. cond. support   (default: sort)
         (only for algorithm variant c, option -Ac)
-u       do not use head union tail (hut) pruning (default: use hut)
         (only for maximal item sets, option -tm)
-F#:#..  support border for filtering item sets   (default: none)
         (list of minimum support values, one per item set size,
         starting at the minimum size, as given with option -m#)
-R#      read item selection/appearance indicators
-P#      write a pattern spectrum to a file
-Z       print item set statistics (number of item sets per size)
-N       do not pre-format some integer numbers   (default: do)
-g       write item names in scanable form (quote certain characters)
-h#      record header  for output               (default: "")
-k#      item separator for output               (default: " ")
-I#      implication sign for association rules   (default: " <- ")
-v#      output format for set/rule information   (default: " (%S)")
-w       integer transaction weight in last field (default: only items)
-r#      record/transaction separators           (default: "\n")
-f#      field /item       separators             (default: " \t,")
-b#      blank   characters                       (default: " \t\r")
-C#      comment characters                       (default: "#")
-!       print additional option information
infile   file to read transactions from           [required]
outfile  file to write item sets/assoc. rules to  [optional]
```

In [32]:
```
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");
```

```
./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 [0.10s].
0 secs  366162 microsecs
```

Answer: FP Growth took the least amount of time because it uses a projection based approach which saves more memory space compared to eclat and apriori

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 [33]:
```
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 [0.01s].
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 [1.28s].
writing <null> ... [29501 set(s)] done [0.00s].
1 secs  550127 microsecs
```

In [34]:
```
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");
```

```
./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 [0.01s].
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 [0.03s].
0 secs  291594 microsecs
```

In [35]:
```
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");
```

```
./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 [0.02s].
0 secs  278357 microsecs
```

Answer: Even after some extra pruning over apriori and slight improvement of eclat, fpgrowth is still the fastest, as it's projection based approach is still fastest and saves more computation time than others

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 [43]:
```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-4=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
./apriori: unknown option -=
0 secs  244224 microsecs
```

In [42]:
```
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");
```

```
./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 [0.00s].
0 secs  271888 microsecs
```

In [41]:
```
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");
```

```
./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 [0.01s].
filtering, sorting and recoding items ... [20 item(s)] done [0.01s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.00s].
writing <null> ... [2254 set(s)] done [0.00s].
0 secs  272033 microsecs
```

Answer: With increase in minsup, this pruning of search space causes apriori to perform better than eclat and fpgrowth. Interstingly FP growth's and Eclat's efficiency doesn't change significantly

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	17.5	1.50	0.24
Eclat	0.53	0.31	0.271
FPGrowth	0.34	0.29	0.272

Answer: As minsup increases the the efficiency of Apriori significatly improves suggesting that the extra pruning of space from a higher minsup has caused this increase, whereas the performance of Eclat and FP Growth doesn't differ significantly

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: We can use substring mining approach to determine the frequently taken paths as they're traversed consecutively (i.e one after another without any gaps or interruptions)

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

In [26]:
```
!wc -l road_seq_data.dat
```

```
1000 road_seq_data.dat
```

Answer:1000

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

In [30]:
```
!awk -- '{for (i = 1; i <= NF; i++) wc[$i] += 1}; END {print length(wc)}' road_se
```

```
1283
```

Answer: 1283

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

Answer:|E| x |E| = 1283 x 1283 = 1646089

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

Answer:|E| x |E| = 1283 x 1283 = 1646089

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

In [31]:
```
!./prefixspan -min_sup 10 road_seq_data.dat| sed -n 'p;n' > road_seq_data_S_10.tx
```

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

In [33]:
```
! wc -l road_seq_data_S_10.txt
```

```
4589 road_seq_data_S_10.txt
```

Answer: 4589

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

In [34]:
```
!chmod u+x seqwog
```

In [35]:
```
!./seqwog
```

```
usage: ./seqwog [options] infile [outfile]
find frequent sequences without gaps
version 3.16 (2016.10.15)        (c) 2010-2016   Christian Borgelt
-t#      target type                             (default: s)
         (s: frequent, c: closed, m: maximal sequences, r: rules)
         (target type 'r' implies -a (all occurrences))
-m#      minimum number of items per sequence    (default: 1)
-n#      maximum number of items per sequence    (default: no limit)
-s#      minimum support of a sequence           (default: 10%)
         (positive: percentage, negative: absolute number)
-o       use original rule support definition    (body & head)
-c#      minimum confidence of a     rule        (default: 80%)
-a       count all occurrences of a pattern      (default: #sequences)
-F#:#..  support border for filtering item sets  (default: none)
         (list of minimum support values, one per item set size,
         starting at the minimum size, as given with option -m#)
-P#      write pattern spectrum to a file
-Z       print item set statistics (number of item sets per size)
-g       write output in scanable form (quote certain characters)
-h#      record header  for output               (default: "")
-k#      item separator for output               (default: " ")
-I#      implication sign for sequence rules     (default: " -> ")
-v#      output format for sequence information  (default: " (%S)")
-w       integer transaction weight in last field (default: only items)
-r#      record/transaction separators           (default: "\n")
-f#      field /item        separators           (default: " \t,")
-b#      blank   characters                      (default: " \t\r")
-C#      comment characters                      (default: "#")
-!       print additional option information
infile   file to read transactions from          [required]
outfile  file to write frequent sequences to     [optional]
```

In [36]:
```
!./seqwog -ts -s-10 road_seq_data.dat substring_result
```

```
./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 [0.00s].
writing substring_result ... [613 sequence(s)] done [0.01s].
```

Answer: 613

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

Answer: Subsequence involves itemsets that contain the subsequence one after another with gaps or interruptions whereas subtring involves itemssets that contains the subsequence without any gaps or interruptions, hence substrings are less in numbers compared to subsequence