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

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 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 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 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 6
3 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 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)}' accidents_
```

```
310
```

Answer: Number of items in the data file accidents_10k are 310

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

In [3]:
```
!wc -l accidents_10k.dat
```

```
10000 accidents_10k.dat
```

Answer: Number of transactions in the data file accidents_10k are 10000.

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

In [1]:
```
!awk -- 'NR==1 || NF<lw {lw=NF} END {print "Length: " lw}' accidents_10k.dat
```

```
Length: 23
```

Answer: Length of smallest transaction in the data file accidents_10k.dat is 23

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

In [5]:
```
!awk -- 'NR==1 || NF>len {len=NF; line=$0} END {print "Length: " len}' accidents_10k
```

```
Length: 45
```

Answer: Length of longest transaction in the data file accidents_10k.dat is 45

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

In [14]:
```
!awk -- '{for (i=1;i<=NF;i++) WC[$i]+=1}; END {printf "Size Space Size %e",2^length(
```

```
Size Space Size 2.085925e+93
```

Answer: The size of search space for frequent itemsets is 2.085925e+93

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: As the data contains thousands of transactions and multiple transactions have multiple items in it, so using the Itemset Mining will help us to prune the subsets from data which are the most important and will allow us to find different relationships between the data. Also, with the help of support for each itemsets, we can discover the most important cause for the accidents and then develop more robut system to reduce the number of accidents due to various factors.

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

Answer: 1. Closed itemset is the best choice for this dataset as it takes less time to generate closed itemset than the frequent itemsets, on other hand it will generate loss-less summary of the itemsets with the support also which is not available in maximal itemsets. 2. Frequent itemset can be second best choice as we will get the most frequent itemsets; we can generte the required association rules with the number of frequent itemsets.

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

Answer: We can assume minsup threshold of 50-70 %, we can also go below 50 % and check whether we can find the occurance of more frequent itemsets. But, as we need to prune the most important itemsets from the data, so we should go ahead with minimum support which is between 50-70%, it will prune the not required data and also provide the important causes of the accidents.

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 [8]:
```
!chmod u+x apriori
```

In [9]:
```
!./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]
```

In [10]:
```
!./apriori -ts -s-2000 accidents_10k.dat A_AP_Freq_minsup2000.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 done [17.36s].
writing A_AP_Freq_minsup2000.txt ... [851034 set(s)] done [0.10s].
```

In [11]:
```
!wc -l A_AP_Freq_minsup2000.txt
```

```
851034 A_AP_Freq_minsup2000.txt
```

In [12]:
```
!./apriori -ts -s-3000 accidents_10k.dat A_AP_Freq_minsup3000.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 ... [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 [4.02s].
writing A_AP_Freq_minsup3000.txt ... [133799 set(s)] done [0.02s].
```

In [13]:
```
!wc -l A_AP_Freq_minsup3000.txt
```

```
133799 A_AP_Freq_minsup3000.txt
```

In [14]:
```
!./apriori -ts -s-4000 accidents_10k.dat A_AP_Freq_minsup4000.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 ... [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 [1.18s].
writing A_AP_Freq_minsup4000.txt ... [29501 set(s)] done [0.00s].
```

In [15]:
```
!wc -l A_AP_Freq_minsup4000.txt
```

```
29501 A_AP_Freq_minsup4000.txt
```

Answer: Firstly, the search space which we have is of 2 raised to power 310. This is a huge space for data analysis. We look for data which is more frequent using the mininum support values. If the minimum support is on lower scale, then it will capture high amount of frequent itemsets and if it is on higher side, then it will capture less number of frequent itemsets. Hence, itemsets with minsup as 2000 generates the highest number of frequent itemsets i.e. 851034 and itemsets with minsup as 4000 generates the lowest number of frequent itemsets i.e. 29501. Apriori prunes superset for that

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 [16]:
```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-2000 accidents_10k.dat A_AP_Freq_minsup2001.txt
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 [0.00s].
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 [17.12s].
writing A_AP_Freq_minsup2001.txt ... [851034 set(s)] done [0.10s].
17 secs  690608 microsecs
```

In [17]:
```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-3000 accidents_10k.dat A_AP_Freq_minsup3001.txt
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 [0.01s].
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 [3.93s].
writing A_AP_Freq_minsup3001.txt ... [133799 set(s)] done [0.02s].
4 secs  247507 microsecs
```

In [18]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-4000 accidents_10k.dat A_AP_Freq_minsup4001.txt
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 [1.17s].
writing A_AP_Freq_minsup4001.txt ... [29501 set(s)] done [0.01s].
1 secs  431825 microsecs
```

Answer: Minsup with value 4000 takes the least amount of time i.e. 1 secs 431825 microsecs, as it will prune the itemsets which are below the support of 4000 at each level(k), which in turn will allow the tree to reach the final leaf node more efficiently.

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 [19]:

```
!awk '{print NF-1}' A_AP_Freq_minsup2000.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 [20]:

```
!awk '{print NF-1}' A_AP_Freq_minsup3000.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 [21]:

```
!awk '{print NF-1}' A_AP_Freq_minsup4000.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: Itemsets with minsup as 2000 are ranging from length 1 to 14. minsup=2000 captures the highest number because the support is on lower side, whereas minsup=3000 and 4000 have less number of lengths of itemsets when compared with minsup=2000 due to obvious reasons that support is high for both of them respectively. Also, the number of itemsets is increasing for first few lengths, but then it is decreasing moving ahead because there may not be large number of frequent itemsets which are of larger lengths.

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 [50]:
```
!./apriori -ts -s-2000 accidents_10k.dat A_AP_Freq_minsup2000.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 [0.03s].
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 [17.51s].
writing A_AP_Freq_minsup2000.txt ... [851034 set(s)] done [0.11s].
```

In [51]:
```
!./apriori -tm -s-2000 accidents_10k.dat A_AP_Max_minsup2000.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 [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 14 done [35.14s].
filtering for maximal item sets ... done [0.05s].
writing A_AP_Max_minsup2000.txt ... [12330 set(s)] done [0.01s].
```

In [52]:
```
!./apriori -tc -s-2000 accidents_10k.dat A_AP_Closed_minsup2000.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 [0.03s].
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 [35.79s].
filtering for closed item sets ... done [0.47s].
writing A_AP_Closed_minsup2000.txt ... [519902 set(s)] done [0.10s].
```

Answer: Frequent itemsets has the highest number of itemsets(851034) and maximal itemsets has lowest number of itemsets(12330). We know that frequent itemset is superset of closed itemset, which in turn is the superset of maximal itemset. So, is the reason that frequent itemsets are the most and maximal itemsets are less in number.

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 [25]:
```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-2000 accidents_10k.dat A_AP_Frequent_minsup2001.txt
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 ... [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 [17.02s].
writing A_AP_Frequent_minsup2001.txt ... [851034 set(s)] done [0.10s].
17 secs  610440 microsecs
```

In [26]:
```
!chmod u+x eclat
```

In [27]:
```
!./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 [28]:

```
import datetime
start = datetime.datetime.now()
!./eclat -ts -s-2000 accidents_10k.dat A_EC_Freq_minsup2000.txt
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 A_EC_Freq_minsup2000.txt ... [851034 set(s)] done [0.34s].
0 secs  842709 microsecs
```

In [29]:

```
!chmod u+x fpgrowth
```

In [30]:

```
!./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 [31]:

```
import datetime
start = datetime.datetime.now()
!./fpgrowth -ts -s-2000 accidents_10k.dat A_FP_Freq_minsup2000.txt
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 [0.03s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.01s].
writing A_FP_Freq_minsup2000.txt ... [851034 set(s)] done [0.15s].
0 secs  644124 microsecs
```

Answer: FP growth algorithm takes the least amount of time to run as it is based on projection based approach to calculate the frequent itemsets. It scans the database only once and works on prefix-span based algorithm so size of projecttion tree shrinks gradually.

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 [32]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-4000 accidents_10k.dat A_AP_Frequent_minsup4001.txt
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 [1.23s].
writing A_AP_Frequent_minsup4001.txt ... [29501 set(s)] done [0.00s].
1 secs  524052 microsecs
```

In [33]:

```
import datetime
start = datetime.datetime.now()
!./eclat -ts -s-4000 accidents_10k.dat A_EC_Freq_minsup4000.txt
```

```
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 ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.01s].
writing A_EC_Freq_minsup4000.txt ... [29501 set(s)] done [0.04s].
0 secs  314034 microsecs
```

In [34]:

```
import datetime
start = datetime.datetime.now()
!./fpgrowth -ts -s-4000 accidents_10k.dat A_FP_Freq_minsup4000.txt
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 A_FP_Freq_minsup4000.txt ... [29501 set(s)] done [0.02s].
0 secs  294505 microsecs
```

Answer: FP growth algorithm takes the least amount of time to run as it is based on projection based approach to calculate the frequent itemsets. It scans the database only once and works on prefix-span based algorithm so size of projecttion tree shrinks gradually.

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 [35]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-6000 accidents_10k.dat A_AP_Frequent_minsup6001.txt
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 ... [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 A_AP_Frequent_minsup6001.txt ... [2254 set(s)] done [0.00s].
0 secs  276758 microsecs
```

In [36]:

```
import datetime
start = datetime.datetime.now()
!./eclat -ts -s-6000 accidents_10k.dat A_EC_Freq_minsup6000.txt
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 A_EC_Freq_minsup6000.txt ... [2254 set(s)] done [0.01s].
0 secs  284987 microsecs
```

In [37]:

```
import datetime
start = datetime.datetime.now()
!./fpgrowth -ts -s-6000 accidents_10k.dat A_FP_Freq_minsup6000.txt
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 ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.00s].
writing A_FP_Freq_minsup6000.txt ... [2254 set(s)] done [0.01s].
0 secs  288566 microsecs
```

Answer: FP growth algorithm takes the least amount of time to run as it is based on projection based approach to calculate the frequent itemsets. It scans the database only once and works on prefix-span based algorithm so size of projecttion tree shrinks gradually.

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	22 secs 461071 microsecs	0 secs 844267 microsecs	0 secs 760493 microsecs
Eclat	2 secs 815011 microsecs	0 secs 346730 microsecs	0 secs 336399 microsecs
FPGrowth	0 secs 306495 microsecs	0 secs 276695 microsecs	0 secs 271429 microsecs

Answer: It's noticed that the execution times of FP growth takes least amount of time beacuse it finds the frequent itemsets in the database without candiate generation and it follows prefix-span algorithm, so it takes less support computation time. In Eclat ,we can see it improves the support computation but not better when compared to the Fp growth as it is scanning dataset at each level. In Apriori , we can see the computation time is higher because it does candidate generation and support computation for each itemset takes the whole database scan, so it takes the most computation time.

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 because order is the main concern here while traversing the paths. Subsequence mining will allow gaps, but substring will not allow gaps.

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

In [38]:

```
!chmod u+x prefixspan
```

In [39]:
```
!./prefixspan
```

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


Usage: prefixspan [OPTION]... INFILE

       where [OPTION]...  is a list of zero or more optional arguments
             INFILE(s)    is the name of the input transaction database

Additional arguments (at most one input file may be specified):
       -min_sup [minimum support]
       -max_pat [maximum pattern]
```

In [1]:
```
!wc -l road_seq_data.dat
```

```
1000 road_seq_data.dat
```

Answer: There are 1000 sequences in road_seq_data.dat file.

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

In [9]:
```
!awk -- '{for (i = 1; i <= NF; i++) wc[$i] += 1}; END {print length(wc)}' road_seq_d
```

```
1283
```

Answer: The size of alphabet is 1283 in the road_seq_data.dat database.

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

In [1]:
```
!./prefixspan -min_sup 1 road_seq_data.dat | sed -n 'p;n' > subsequence.dat
```

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

In [2]:
```
!awk 'NF==2 { count++} END {print count}' subsequence.dat
```

```
15927
```

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

In [43]:
```
!chmod u+x seqwog
```

In [44]:
```
!./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 [14]:
```
!./seqwog -ts -s-2 -n2 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 ... [883/1000 transaction(s)] done [0.00s].
writing substring_result ... [1390 sequence(s)] done [0.01s].
```

In [15]:
```
!wc -l substring_result
```

```
1390 substring_result
```

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

In [5]:
```
!./prefixspan -min_sup 10 road_seq_data.dat | sed -n 'p;n' > roadseq_Freq_10
```

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

In [6]:
```
!wc -l roadseq_Freq_10
```

```
4589 roadseq_Freq_10
```

Answer: There are 4589 frequent subsequences for minsup = 10.

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

In [48]:
```
!./seqwog -ts -s-10 road_seq_data.dat substring_result_10
```

```
./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_10 ... [613 sequence(s)] done [0.00s].
```

In [49]:
```
!wc -l substring_result_10
```

```
613 substring_result_10
```

Answer: There are 613 frequent substrings with minsup = 10.

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

Answer: There is a huge difference in the subsequence and substrings because the substrings will not allow gaps to be considered from a particular sequence, but subsequence are consecutive in form and they eliminate multiple subsequences.