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 63 66 80 81 82 8
3 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 [1]:

```
!awk -- '{for (i = 1; i <= NF; i++) wc[$i] += 1}; END {print length(wc)}' accidents_10k.dat
```

```
310
```

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

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

In [2]:

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

```
10000 accidents_10k.dat
```

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

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

In [41]:

```
!awk '{print NF}' accidents_10k.dat|sort -n|uniq -c
```

```
      4 23
      5 24
      5 25
     23 26
     52 27
    122 28
    326 29
    739 30
    883 31
   1023 32
   1240 33
   1388 34
   1256 35
```

```
1082 36
 760 37
 503 38
 291 39
 162 40
  78 41
  35 42
  12 43
   6 44
   5 45
```

Answer: There are four transcations with length 23 ,so length of the smallest transaction is 23

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

In [42]:

```
!awk '{print NF}' accidents_10k.dat|sort -nr|uniq -c
```

```
   5 45
   6 44
  12 43
  35 42
  78 41
 162 40
 291 39
 503 38
 760 37
1082 36
1256 35
1388 34
1240 33
1023 32
 883 31
 739 30
 326 29
 122 28
  52 27
  23 26
   5 25
   5 24
   4 23
```

Answer: There are five transactions with lenght 45 ,So the length of longest transaction is 45

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

Answer: The size of the search space is 2^(number of items) ,So 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: By using itemset mining we can find the relationship between different attributes that are causing road accidents frequently.By knowing the relationship we can try and take measures to prevent the accidents if the same set of attributes repeat.

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

Answer: I would be interested in discovering Maximal itemsets because maximal itemsets also covers frequent itemsets,that is,subsets of maximal itemset will also be frequent.Moreover, the supersets will be less frequent,so we dont have to spend time in finding supersets.

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

Answer: I would choose a relative minsup value rather than an absolute value .I would like to have it as 80% of size of data base because from question 1d we can say that most of the transactions which are long are covered in the 80%.

2. Generating frequent, maximal and closed itemsets using $\color{red}{\text{Apriori}}$, $\color{red}{\text{ECLAT}}$, and $\color{red}{\text{FPGrowth}}$ 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 [2]:

```
!chmod u+x apriori
```

In [13]:

```
!./apriori -ts -s-2000 accidents_10k.dat ap_Freq_2k.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 [18.72s].
writing ap_Freq_2k.txt ... [851034 set(s)] done [0.10s].
```

In [14]:

```
!./apriori -ts -s-3000 accidents_10k.dat ap_Freq_3k.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 [4.32s].
writing ap_Freq_3k.txt ... [133799 set(s)] done [0.01s].
```

In [15]:

```
!./apriori -ts -s-4000 accidents_10k.dat ap_Freq_4k.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 [1.13s].
writing ap_Freq_4k.txt ... [29501 set(s)] done [0.00s].
```

Answer: For minsup=2000 we can observe the maximum number of frequent itemsets, that is,851034. For minsup=4000 we can observe minimum number of frequent itemsets,that is,29501. As the minsup value increases the number of frequent itemsets decreases

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 [6]:

```
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 [0.01s].
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 [18.66s].
writing <null> ... [851034 set(s)] done [0.01s].
19 secs  320856 microsecs
```

In [7]:

```
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 [0.01s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.00s].
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 [4.37s].
writing <null> ... [133799 set(s)] done [0.00s].
4 secs  748388 microsecs
```

In [8]:

```
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 [1.28s].
writing <null> ... [29501 set(s)] done [0.00s].
1 secs  587114 microsecs
```

Answer: Minsup =4000 takes the least time.As the minsup increases the search space decreases ,so execution time decreases.

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 [32]:

```
!awk '{print NF-1}' ap_Freq_2k.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}' ap_Freq_3k.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}' ap_Freq_4k.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: =>The length of the frequent itemset decreased as the minsup value increased. =>The values of length of the frequent itemsets is following a normal distribution for all the minsup values. =>The number of frequent itemsets decreased as the value of minsup increased.

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 -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 [0.00s].
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 [19.30s].
writing <null> ... [851034 set(s)] done [0.01s].
```

In [22]:

```
!./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 [0.02s].
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 [29.92s].
filtering for maximal item sets ... done [0.03s].
```

```
filtering for maximal item sets ... done [0.05s].
writing <null> ... [12330 set(s)] done [0.02s].
```

In [23]:

```
!./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 [0.00s].
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 [28.30s].
filtering for closed item sets ... done [0.47s].
writing <null> ... [519902 set(s)] done [0.01s].
```

Answer: The largest set is frequent itemsets and the smallest set is maximal itemsets.The supersets of maximal itemsets will not be frequent but the subsets of maximal itemsets will be frequent.So there will be a large number of frequent itemsets and comaparitively less number of maximal 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 [24]:

```
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 [0.01s].
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 [18.72s].
writing <null> ... [851034 set(s)] done [0.02s].
19 secs  370660 microsecs
```

In [26]:

```
!chmod u+x eclat
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 [0.27s].
0 secs  610845 microsecs
```

In [27]:

```
!chmod u+x fpgrowth
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 [0.00s].
writing <null> ... [851034 set(s)] done [0.09s].
0 secs  378623 microsecs
```

Answer: fpgrowth took least amount of time.In apriori the database scan happen multiple times so it takes lot of time to execute when compared with fpgrowth.In fpgrowth the time complexity is reduced because you have a projected database which is shrinked when compared with the Eclat.

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 [29]:

```
!chmod u+x apriori
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 [0.01s].
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 [1.28s].
writing <null> ... [29501 set(s)] done [0.00s].
1 secs  601481 microsecs
```

In [30]:

```
!chmod u+x eclat
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 [0.01s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.00s].
writing <null> ... [29501 set(s)] done [0.04s].
0 secs  323315 microsecs
```

In [32]:

```
!chmod u+x fpgrowth
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 [0.01s].
writing <null> ... [29501 set(s)] done [0.02s].
```

```
0 secs  299456 microsecs
```

Answer: fpgrowth took least time to execute.since the minsup value is increased the time to execute reduces for all the algorithms and this is more evident in apriori as the number of times of database scan reduces.

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 [33]:

```
!chmod u+x apriori
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");
```

```
./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 [0.01s].
building transaction tree ... [6478 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 done [0.03s].
writing <null> ... [2254 set(s)] done [0.00s].
0 secs  322153 microsecs
```

In [36]:

```
!chmod u+x eclat
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 [0.01s].
filtering, sorting and recoding items ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.01s].
writing <null> ... [2254 set(s)] done [0.00s].
0 secs  285174 microsecs
```

In [37]:

```
!chmod u+x fpgrowth
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 [0.02s].
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  282904 microsecs
```

Answer: fpgrowth took least time to execute.As the number of frequent itemsets are very less for high minsup value ,the shrink in database is also less.So the time difference between Eclat and fpgrowth is much less.

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	19 secs 370660 microsecs	1 secs 601481 microsecs	322153 microsecs
Eclat	0 secs 610845 microsecs	323315 microsecs	285174 microsecs
FPGrowth	0 secs 378623 microsecs	299456 microsecs	282904 microsecs

Answer: FPGrowth is the best for the given minsup values.As the minsup value increase the reduction in time complexity is more evident in apriori compared to Eclat and FPgrowth as the -number of database scans reduces.

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: I would choode seqwog because the substrings are continuous in nature ,so they provide us better information in identifying the sequences of paths.

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

In [16]:

```
!chmod u+x prefixspan
!wc -l road_seq_data.dat
```

```
1000 road_seq_data.dat
```

Answer: 1000 sequences are present in this sequence database

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

In [47]:

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

```
1283
```

Answer: The size is 1283

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

In [28]:

```
!awk '{for (k=1;k<NF;k++) sum=sum+NF-k}; END {print sum}' road_seq_data.dat
# ∑_(k=1)^(n-1)[(n-k)] this formula works only for subsequence length of 2.
```

```
46453
```

Answer: There are 46453 subsequences of length 2

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

In [19]:

```
!awk '{print NF-2+1}' road_seq_data.dat >substr_len.txt
!awk '{ sum += $1 } END { print sum }' substr_len.txt
#NF-L+1 works, where L is the length of substring
```

```
8940
```

Answer: There 8940 substrings of length 2

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

In [11]:

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

In [33]:

```
!./prefixspan -min_sup 10 road_seq_data.dat|sed -n 'p;n' > subseq_min10.txt
```

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

In [34]:

```
!wc -l subseq_min10.txt
```

```
4589 subseq_min10.txt
```

Answer: 4589 subsequences are disovered for minsup=10

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

In [9]:

```
!chmod u+x seqwog
```

In [10]:

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

```
./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 <null> ... [613 sequence(s)] done [0.01s].
```

Answer: 613 substrings have been discovered

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

Answer: In a given sequence the subsequences are not continuous whereas substrings should be continuous ,so the number of subsequences is large compared to substrings