Running different algorithms for sequential patterns

- **Aprioriall**
- **PrefixSpan (Python and Java implementations)**

The Aprioriall algorithm

In [1]:
```
1 import aprioriall
```

The format of the dataset is:

- An event is a list of strings.
- A sequence is a list of events.
- A dataset is a list of sequences.

Example:

```
dataset =  [
      [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
      [["a"], ["c"], ["b", "c"]],
      [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
      [["a"], ["c"], ["b"], ["c"]]
]
```

In [2]:

```
dataset =  [
    [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
    [["a"], ["c"], ["b", "c"]],
    [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
    [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
    [["a"], ["c"], ["b", "c"]],
    [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
    [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
    [["a"], ["c"], ["b", "c"]],
    [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
    [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
    [["a"], ["c"], ["b", "c"]],
    [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
    [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
    [["a"], ["c"], ["b", "c"]],
    [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
    [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
    [["a"], ["c"], ["b", "c"]],
    [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
    [["a"], ["a", "b", "c"], ["a", "c"], ["c"]],
    [["a"], ["c"], ["b", "c"]],
    [["a", "b"], ["d"], ["c"], ["b"], ["c"]],
    [["a"], ["c"], ["b"], ["c"]]
]
```

In [3]:

```
dataset =  [
    [["a"], ["b", "c"], ["d"]],
    [["e"], ["a"], ["e"], ["b","c"], ["f"], ["d"]],
    [["h"], ["h", "i"], ["j"]],
    [["h"], ["i"], ["j"], ["k"]]
]

result = aprioriall.apriori(dataset, 2, verbose=False)
aprioriall.filterMaximal(result)
print(result)
result = aprioriall.apriori(dataset, 2, verbose=False)
aprioriall.filterClosed(result)
print(result)
```

```
Result, lvl 1: [([['a']], 2), ([['b']], 2), ([['c']], 2), ([['d']], 2), ([['h']], 2), ([['i']], 2), ([['j']], 2)]
[([['h'], ['i'], ['j']], 2), ([['a'], ['b', 'c'], ['d']], 2)]
Result, lvl 1: [([['a']], 2), ([['b']], 2), ([['c']], 2), ([['d']], 2), ([['h']], 2), ([['i']], 2), ([['j']], 2)]
[([['h'], ['i'], ['j']], 2), ([['a'], ['b', 'c'], ['d']], 2)]
```

Running aprioriall: aprioriall.apriori (nameofthedataset, support=number of minimal occurrences, verbose={false/true})

In [4]:
```
1  apprioriall.apriori(dataset, 2, verbose=False)
```

```
Result, lvl 1: ([['a']], 2), ([['b']], 2), ([['c']], 2), ([['d']],
2), ([['h']], 2), ([['i']], 2), ([['j']], 2)]
```

Out[4]:
```
[([['a']], 2),
 ([['b']], 2),
 ([['c']], 2),
 ([['d']], 2),
 ([['h']], 2),
 ([['i']], 2),
 ([['j']], 2),
 ([['b', 'c']], 2),
 ([['a'], ['b']], 2),
 ([['a'], ['c']], 2),
 ([['a'], ['d']], 2),
 ([['b'], ['d']], 2),
 ([['c'], ['d']], 2),
 ([['h'], ['i']], 2),
 ([['h'], ['j']], 2),
 ([['i'], ['j']], 2),
 ([['a'], ['b'], ['d']], 2),
 ([['a'], ['b', 'c']], 2),
 ([['a'], ['c'], ['d']], 2),
 ([['b', 'c'], ['d']], 2),
 ([['h'], ['i'], ['j']], 2),
 ([['a'], ['b', 'c'], ['d']], 2)]
```

Get the maximal sequential patterns

In [5]:
```
1  result = apprioriall.apriori(dataset, 2, verbose=False)
2  apprioriall.filterMaximal(result)
3  print(result)
```

```
Result, lvl 1: ([['a']], 2), ([['b']], 2), ([['c']], 2), ([['d']],
2), ([['h']], 2), ([['i']], 2), ([['j']], 2)]
[([['h'], ['i'], ['j']], 2), ([['a'], ['b', 'c'], ['d']], 2)]
```

Get the closed sequential patterns

In [6]:
```
1  result = apprioriall.apriori(dataset, 2, verbose=False)
2  apprioriall.filterClosed(result)
3  print(result)
```

```
Result, lvl 1: ([['a']], 2), ([['b']], 2), ([['c']], 2), ([['d']],
2), ([['h']], 2), ([['i']], 2), ([['j']], 2)]
[([['h'], ['i'], ['j']], 2), ([['a'], ['b', 'c'], ['d']], 2)]
```

The PrefixSpan algorithm

In [7]:
```
import prefixspan
```

Running aprioriall: prefixspan.prefixSpan(nameofthedataset, support=number of minimal occurrences)

In [8]:
```
prefixspan.prefixSpan(dataset, 2)
```

Out[8]:
```
[([['a']], 2),
 ([['a'], ['b']], 2),
 ([['a'], ['b', 'c']], 2),
 ([['a'], ['b', 'c'], ['d']], 2),
 ([['a'], ['b'], ['d']], 2),
 ([['a'], ['c']], 2),
 ([['a'], ['c'], ['d']], 2),
 ([['a'], ['d']], 2),
 ([['b']], 2),
 ([['b', 'c']], 2),
 ([['b', 'c'], ['d']], 2),
 ([['b'], ['d']], 2),
 ([['c']], 2),
 ([['c'], ['d']], 2),
 ([['d']], 2),
 ([['h']], 2),
 ([['h'], ['i']], 2),
 ([['h'], ['i'], ['j']], 2),
 ([['h'], ['j']], 2),
 ([['i']], 2),
 ([['i'], ['j']], 2),
 ([['j']], 2)]
```

Get the maximal sequential patterns

In [9]:
```
result = prefixspan.prefixSpan (dataset, 2)
prefixspan.filterMaximal(result)
print(result)
```

```
[([['a'], ['b', 'c'], ['d']], 2), ([['h'], ['i'], ['j']], 2)]
```

Get the closed sequential patterns

In [10]:
```
result = prefixspan.prefixSpan (dataset, 2)
prefixspan.filterClosed(result)
print(result)
```

```
[([['a'], ['b', 'c'], ['d']], 2), ([['h'], ['i'], ['j']], 2)]
```

Comparing execution times

In [11]:
```
print ('Time for Aprioriall\n')
%time aprioriall.apriori(dataset, 2, verbose=False)
print ('\n\nTime for prefixspan\n')
%time prefixspan.prefixSpan (dataset, 2)
```

```
Time for Aprioriall

Result, lvl 1: ([['a']], 2), ([['b']], 2), ([['c']], 2), ([['d']],
2), ([['h']], 2), ([['i']], 2), ([['j']], 2)]
CPU times: user 8.02 ms, sys: 1.07 ms, total: 9.09 ms
Wall time: 8.6 ms


Time for prefixspan

CPU times: user 556 µs, sys: 0 ns, total: 556 µs
Wall time: 561 µs
```

Out[11]:
```
[([['a']], 2),
 ([['a'], ['b']], 2),
 ([['a'], ['b', 'c']], 2),
 ([['a'], ['b', 'c'], ['d']], 2),
 ([['a'], ['b'], ['d']], 2),
 ([['a'], ['c']], 2),
 ([['a'], ['c'], ['d']], 2),
 ([['a'], ['d']], 2),
 ([['b']], 2),
 ([['b', 'c']], 2),
 ([['b', 'c'], ['d']], 2),
 ([['b'], ['d']], 2),
 ([['c']], 2),
 ([['c'], ['d']], 2),
 ([['d']], 2),
 ([['h']], 2),
 ([['h'], ['i']], 2),
 ([['h'], ['i'], ['j']], 2),
 ([['h'], ['j']], 2),
 ([['i']], 2),
 ([['i'], ['j']], 2),
 ([['j']], 2)]
```

Dealing with real datasets

Many datasets are available in the following URL to deal with sequential patterns. They are in a specific format called SPMF :
http://www.philippe-fournier-viger.com/spmf (http://www.philippe-fournier-viger.com/spmf)

In [12]:

```
import re
import urllib.request
# Mapping function to transform SPMF data
# this function allows to use a local file in SPMF or an URL
def SPMFFormatConverter(name):
    sequences = []
    if re.match(name,"http:") == None:
        u = urllib.request.urlopen('http://www.philippe-fournier
        f = u.readlines()
    else:
        f = open(name)
    for ln in f:
        if re.match(name,"http:") == None:
            seq = ln.decode('utf8').split(" -1 ")
        else:
            seq = ln.split(" -1 ")
        # remove end of line : "-2\n" and last line "-2"
        seq = [x for x in seq if (x != "-2\n" and x != "-2")]
        sequences.append(seq)
    if re.match(name,"http:") != None:
        f.close()

    #Create sequences of items
    Data=[]
    for seq in sequences:
        newSeq = []
        for item in seq:
            newSeq.append([item])
        Data.append(newSeq)
    return Data

```

An example:
BMSWebView1 (Gazelle) (KDD CUP 2000). This dataset contains 59,601 sequences of clickstream data from an e-commerce. It contains 497 distinct items. The average length of sequences is 2.42 items with a standard deviation of 3.22. In this dataset, there are some long sequences. For example, 318 sequences contains more than 20 items. The dataset is available here: http://www.philippe-fournier-viger.com/spmf/datasets/BMS1_spmf (http://www.philippe-fournier-viger.com/spmf/datasets/BMS1_spmf)

In [13]:
```
#local file
LogData=SPMFFormatConverter('BMS1_spmf1.txt')


LogData=[]
#using url
LogData=SPMFFormatConverter('http://www.philippe-fournier-viger.
print ("\n The ten first sequences of clickstreams\n")
print (LogData[1:10])
print ('Number of sequences:',len(LogData))

```

```
 The ten first sequences of clickstreams

[[['12559']], [['12695'], ['12703'], ['18715']], [['10311'], ['12387
'], ['12515'], ['12691'], ['12695'], ['12699'], ['12703'], ['12823']
, ['12831'], ['12847'], ['18595'], ['18679'], ['18751']], [['10291']
, ['12523'], ['12531'], ['12535'], ['12883']], [['12523'], ['12539']
, ['12803'], ['12819']], [['12819']], [['12471'], ['12491'], ['12531
'], ['12535'], ['12567'], ['12663'], ['12667'], ['12823'], ['18447']
, ['18507'], ['18691']], [['12487']], [['12547'], ['12815'], ['12895
']]]
Number of sequences: 59601
```

In [14]:
```
%time result=prefixspan.prefixSpan (LogData, 200)

sorted(result)
```

```
CPU times: user 39.6 s, sys: 343 ms, total: 39.9 s
Wall time: 41.8 s
```

Out[14]:
```
[(([['10291']], 472),
 (([['10295']], 2009),
 (([['10295'], ['10299']], 323),
 (([['10295'], ['10299'], ['10307']], 297),
 (([['10295'], ['10307']], 916),
 (([['10295'], ['10307'], ['10311']], 417),
 (([['10295'], ['10307'], ['10311'], ['10315']], 205),
 (([['10295'], ['10307'], ['10315']], 335),
 (([['10295'], ['10307'], ['12695']], 213),
 (([['10295'], ['10307'], ['12895']], 246),
 (([['10295'], ['10311']], 738),
 (([['10295'], ['10311'], ['10315']], 351),
 (([['10295'], ['10315']], 722),
 (([['10295'], ['12483']], 227),
 (([['10295'], ['12487']], 266),
 (([['10295'], ['12679']], 211),
 (([['10295'], ['12695']], 336),
```

The closed sequential patterns

In [15]:
```
prefixspan.filterClosed(result)
```

```
2 print(result)
```

```
[(([['10291']], 472), ([['10295']], 2009), ([['10295'], ['10299']], 3
23), ([['10295'], ['10299'], ['10307']], 297), ([['10295'], ['10307'
]], 916), ([['10295'], ['10307'], ['10311']], 417), ([['10295'], ['1
0307'], ['10311'], ['10315']], 205), ([['10295'], ['10307'], ['10315
']], 335), ([['10295'], ['10307'], ['12695']], 213), ([['10295'], ['
10307'], ['12895']], 246), ([['10295'], ['10311']], 738), ([['10295'
], ['10311'], ['10315']], 351), ([['10295'], ['10315']], 722), ([['1
0295'], ['12483']], 227), ([['10295'], ['12487']], 266), ([['10295']
, ['12679']], 211), ([['10295'], ['12695']], 336), ([['10295'], ['12
703']], 259), ([['10295'], ['12819']], 202), ([['10295'], ['12827']]
, 201), ([['10295'], ['12831']], 217), ([['10295'], ['12895']], 398)
, ([['10295'], ['33449']], 280), ([['10295'], ['33469']], 228), ([['
10299']], 721), ([['10299'], ['10307']], 460), ([['10299'], ['10307'
], ['10311']], 216), ([['10299'], ['10311']], 238), ([['10303']], 46
1), ([['10307']], 2797), ([['10307'], ['10311']], 621), ([['10307'],
['10311'], ['10315']], 288), ([['10307'], ['10315']], 496), ([['1030
7'], ['12483']], 239), ([['10307'], ['12487']], 263), ([['10307'], [
'12679']], 286), ([['10307'], ['12695']], 281), ([['10307'], ['12703
']], 254), ([['10307'], ['12827']], 212), ([['10307'], ['12831']], 2
33), ([['10307'], ['12895']], 439), ([['10307'], ['33449']], 228), (
[['10307'], ['33469']], 242), ([['10311']], 2371), ([['10311'], ['10
315']], 771), ([['10311'], ['10315'], ['12487']], 227), ([['10311'],
['10315'], ['12703']], 204), ([['10311'], ['12483']], 469), ([['1031
1'], ['12483'], ['12487']], 310), ([['10311'], ['12487']], 615), ([[
'10311'], ['12487'], ['12703']], 322), ([['10311'], ['12487'], ['127
03'], ['32213']], 200), ([['10311'], ['12487'], ['12875']], 222), ([
['10311'], ['12487'], ['32213']], 256), ([['10311'], ['12695']], 256
), ([['10311'], ['12703']], 576), ([['10311'], ['12703'], ['32213']]
, 295), ([['10311'], ['12875']], 337), ([['10311'], ['12895']], 281)
, ([['10311'], ['32213']], 463), ([['10311'], ['34893']], 246), ([['
10315']], 3449), ([['10315'], ['10331']], 252), ([['10315'], ['10335
']], 426), ([['10315'], ['10339']], 242), ([['10315'], ['12483']], 3
56), ([['10315'], ['12487']], 381), ([['10315'], ['12679']], 225), (
[['10315'], ['12695']], 233), ([['10315'], ['12703']], 330), ([['103
15'], ['12831']], 225), ([['10315'], ['12875']], 208), ([['10315'],
['12895']], 356), ([['10315'], ['32213']], 274), ([['10315'], ['3344
9']], 228), ([['10327']], 232), ([['10331']], 690), ([['10331'], ['1
0335']], 235), ([['10335']], 1167), ([['10339']], 455), ([['10829']]
, 207), ([['10833']], 310), ([['10841']], 328), ([['10849']], 318),
([['10857']], 448), ([['10857'], ['10861']], 204), ([['10861']], 469
), ([['10865']], 295), ([['10877']], 1389), ([['10881']], 389), ([['
12299']], 562), ([['12327']], 294), ([['12331']], 208), ([['12335']]
, 204), ([['12339']], 910), ([['12339'], ['35185']], 251), ([['12343
']], 267), ([['12347']], 275), ([['12355']], 1089), ([['12355'], ['1
8863']], 207), ([['12367']], 383), ([['12371']], 240), ([['12395']],
294), ([['12399']], 320), ([['12403']], 411), ([['12407']], 756), ([
['12411']], 797), ([['12419']], 331), ([['12427']], 211), ([['12431'
]], 1198), ([['12431'], ['18863']], 245), ([['12439']], 296), ([['12
447']], 388), ([['12451']], 314), ([['12459']], 345), ([['12463']],
797), ([['12463'], ['12487']], 204), ([['12467']], 474), ([['12471']
], 230), ([['12475']], 207), ([['12479']], 731), ([['12479'], ['1248
3']], 290), ([['12479'], ['12487']], 285), ([['12483']], 2049), ([['
```

12483'], ['12487']], 877), ([['12483'], ['12487'], ['12703']], 259), ([['12483'], ['12487'], ['12875']], 228), ([['12483'], ['12703']], 337), ([['12483'], ['12875']], 290), ([['12483'], ['32213']], 273), ([['12483'], ['34893']], 215), ([['12487']], 2268), ([['12487'], ['12571']], 247), ([['12487'], ['12695']], 256), ([['12487'], ['12695'], ['12703']], 204), ([['12487'], ['12703']], 631), ([['12487'], ['12703'], ['12875']], 216), ([['12487'], ['12703'], ['32213']], 315), ([['12487'], ['12703'], ['34893']], 225), ([['12487'], ['12875']], 432), ([['12487'], ['12895']], 226), ([['12487'], ['32213']], 506), ([['12487'], ['34893']], 351), ([['12491']], 440), ([['12495']], 543), ([['12515']], 410), ([['12523']], 449), ([['12527']], 463), ([['12531']], 410), ([['12547']], 471), ([['12551']], 527), ([['12555']], 477), ([['12559']], 1099), ([['12563']], 242), ([['12567']], 890), ([['12571']], 676), ([['12571'], ['12575']], 246), ([['12571'], ['12703']], 222), ([['12571'], ['32213']], 212), ([['12575']], 485), ([['12579']], 244), ([['12583']], 229), ([['12587']], 223), ([['12591']], 264), ([['12595']], 219), ([['12603']], 648), ([['12607']], 365), ([['12611']], 587), ([['12611'], ['12621']], 225), ([['12615']], 237), ([['12621']], 1488), ([['12627']], 781), ([['12631']], 293), ([['12643']], 251), ([['12647']], 411), ([['12655']], 758), ([['12659']], 1006), ([['12663']], 1793), ([['12663'], ['12667']], 269), ([['12663'], ['12679']], 239), ([['12663'], ['12895']], 258), ([['12667']], 570), ([['12671']], 297), ([['12675']], 553), ([['12679']], 1788), ([['12679'], ['12683']], 326), ([['12679'], ['12695']], 256), ([['12679'], ['12703']], 385), ([['12679'], ['12895']], 411), ([['12683']], 779), ([['12683'], ['12691']], 214), ([['12683'], ['12695']], 285), ([['12683'], ['12703']], 248), ([['12683'], ['12895']], 203), ([['12687']], 724), ([['12687'], ['12691']], 207), ([['12691']], 1002), ([['12691'], ['12695']], 297), ([['12691'], ['12703']], 237), ([['12691'], ['33449']], 201), ([['12695']], 1422), ([['12695'], ['12703']], 615), ([['12695'], ['12895']], 266), ([['12695'], ['33449']], 267), ([['12703']], 1948), ([['12703'], ['12875']], 303), ([['12703'], ['12895']], 252), ([['12703'], ['32213']], 571), ([['12703'], ['34893']], 320), ([['12711']], 521), ([['12715']], 1214), ([['12715'], ['12723']], 237), ([['12719']], 377), ([['12723']], 1267), ([['12723'], ['12727']], 215), ([['12727']], 486), ([['12731']], 345), ([['12735']], 439), ([['12739']], 407), ([['12743']], 462), ([['12747']], 511), ([['12751']], 1039), ([['12751'], ['18863']], 259), ([['12755']], 652), ([['12759']], 845), ([['12763']], 696), ([['12767']], 409), ([['12771']], 671), ([['12775']], 438), ([['12779']], 794), ([['12783']], 945), ([['12787']], 440), ([['12795']], 908), ([['12795'], ['12819']], 307), ([['12795'], ['12831']], 359), ([['12795'], ['12895']], 228), ([['12803']], 273), ([['12807']], 583), ([['12815']], 704), ([['12815'], ['12827']], 202), ([['12815'], ['12895']], 552), ([['12819']], 1113), ([['12819'], ['12831']], 213), ([['12819'], ['12895']], 214), ([['12823']], 605), ([['12827']], 1017), ([['12827'], ['12831']], 202), ([['12827'], ['12895']], 590), ([['12831']], 1180), ([['12831'], ['12895']], 250), ([['12835']], 533), ([['12839']], 256), ([['12843']], 396), ([['12847']], 473), ([['12851']], 209), ([['12855']], 436), ([['12867']], 483), ([['12871']], 379), ([['12875']], 1355), ([['12875'], ['12895']], 257), ([['12875'], ['32213']], 216), ([['12875'], ['34893']], 259), ([['12879']], 410), ([['12883']], 498), ([['12887']], 308), ([['12891']], 209), ([['12895']], 3623), ([['12895'], ['18863']], 246), ([['12895'], ['33449']], 345), ([['12895'], [

```
'33469']], 268), ([['12903']], 351), ([['12907']], 448), ([['18423']
], 347), ([['18427']], 439), ([['18447']], 235), ([['18487']], 218),
([['18499']], 260), ([['18507']], 202), ([['18527']], 300), ([['1854
7']], 407), ([['18587']], 221), ([['18603']], 230), ([['18619']], 45
7), ([['18643']], 204), ([['18675']], 234), ([['18683']], 211), ([['
18691']], 467), ([['18707']], 412), ([['18723']], 319), ([['18727']]
, 222), ([['18787']], 736), ([['18831']], 410), ([['18855']], 334),
([['18863']], 863), ([['18871']], 394), ([['18879']], 355), ([['2080
7']], 475), ([['32197']], 280), ([['32201']], 596), ([['32201'], ['3
2205']], 261), ([['32201'], ['32213']], 200), ([['32205']], 871), ([
['32205'], ['32213']], 381), ([['32209']], 516), ([['32213']], 1616)
, ([['32213'], ['34893']], 275), ([['32221']], 233), ([['33425']], 4
05), ([['33429']], 600), ([['33429'], ['33449']], 262), ([['33433']]
, 891), ([['33433'], ['33449']], 424), ([['33433'], ['33449'], ['334
69']], 284), ([['33433'], ['33453']], 305), ([['33433'], ['33469']],
509), ([['33437']], 386), ([['33441']], 260), ([['33449']], 3658), (
[['33449'], ['33453']], 281), ([['33449'], ['33453'], ['33469']], 21
9), ([['33449'], ['33469']], 1204), ([['33449'], ['34885']], 238), (
[['33453']], 651), ([['33453'], ['33469']], 403), ([['33457']], 206)
, ([['33465']], 297), ([['33469']], 3612), ([['34885']], 937), ([['3
4885'], ['34893']], 230), ([['34885'], ['34897']], 289), ([['34889']
], 755), ([['34889'], ['34901']], 248), ([['34889'], ['34905']], 230
), ([['34893']], 1201), ([['34893'], ['34897']], 212), ([['34897']],
682), ([['34901']], 617), ([['34901'], ['34905']], 210), ([['34905']
], 723), ([['34909']], 257), ([['34913']], 291), ([['34917']], 247),
([['34921']], 336), ([['34925']], 278), ([['34929']], 319), ([['3514
9']], 272), ([['35153']], 240), ([['35157']], 212), ([['35165']], 23
2), ([['35169']], 272), ([['35177']], 255), ([['35181']], 290), ([['
35185']], 639), ([['35213']], 358), ([['46281']], 201), ([['47945']]
, 620), ([['47945'], ['47973']], 210), ([['47949']], 512), ([['47953
']], 561), ([['47953'], ['47957']], 251), ([['47957']], 627), ([['47
965']], 329), ([['47973']], 337), ([['48575']], 241), ([['48663']],
284), ([['48663'], ['48667']], 217), ([['48667']], 491), ([['48667']
, ['48675']], 212), ([['48675']], 587), ([['48707']], 335), ([['5466
1']], 251), ([['54673']], 233)]
```

Comparing execution times between Aprioriall and PrefixSpan. The minimal support is 3000 in order to avoid to wait too long.

In [16]:
```
print ('Time for AprioriAll: \n')
%time result = apriori(LogData, 3000, False)

print ('\nTime for PrefixSpan: \n')
%time result = prefixspan.prefixSpan (LogData, 3000)
```

```
Time for AprioriAll:

Result, lvl 1: [([['10315']], 3449), ([['12895']], 3623), ([['33449'
]], 3658), ([['33469']], 3612)]
CPU times: user 1min 27s, sys: 788 ms, total: 1min 28s
Wall time: 1min 31s

Time for PrefixSpan:

CPU times: user 884 ms, sys: 8.47 ms, total: 892 ms
Wall time: 911 ms
```

Using SPMF java code

The SPMF java code can be downloaded here:

* http://www.philippe-fournier-viger.com/spmf/index.php?link=download.php

in the following it is assumed that the jar file is saved in the current directory.
Furthermore, it is assumed that the files used for the test have also been downloaded and saved in the directory:

* Dataset/SPMF/test_files

Finally it is assumed that the file KDDCUP2000BMS1_spmf.txt is in the current directory. According to your configuration, think to update the path for accessing the datasets.

In [17]:
```
def SPMFResultConverter (name):
    f=open("output.txt")
    results=[]
    for ln in f:
        seq = ln.split(" -1 ")
        res='<'
        for x in seq:
            if x.find("SUP") == -1:
                res=res+'['+x+"]"
            else:
                x=x.replace('\n','')
                res=res+'> '+x
        results.append(res)
    return results
```

In [18]:
```
import os
```

In [18]:
```
import os

os.system("java -jar spmf.jar run PrefixSpan Dataset/SPMF/test_f

results=SPMFResultConverter("output.txt")
sorted(results)

```

Out[18]:
```
['<[1 2]> #SUP: 2',
 '<[1 2][3]> #SUP: 2',
 '<[1 2][4]> #SUP: 2',
 '<[1 2][4][3]> #SUP: 2',
 '<[1 2][6]> #SUP: 2',
 '<[1]> #SUP: 4',
 '<[1][1]> #SUP: 2',
 '<[1][2 3]> #SUP: 2',
 '<[1][2 3][1]> #SUP: 2',
 '<[1][2]> #SUP: 4',
 '<[1][2][1]> #SUP: 2',
 '<[1][2][3]> #SUP: 2',
 '<[1][3]> #SUP: 4',
 '<[1][3][1]> #SUP: 2',
 '<[1][3][2]> #SUP: 3',
 '<[1][3][3]> #SUP: 3',
 '<[1][4]> #SUP: 2',
 '<[1][4][3]> #SUP: 2',
 '<[1][6]> #SUP: 2',
 '<[2 3]> #SUP: 2',
 '<[2 3][1]> #SUP: 2',
 '<[2]> #SUP: 4',
 '<[2][1]> #SUP: 2',
 '<[2][3]> #SUP: 3',
 '<[2][4]> #SUP: 2',
 '<[2][4][3]> #SUP: 2',
 '<[2][6]> #SUP: 2',
 '<[3]> #SUP: 4',
 '<[3][1]> #SUP: 2',
 '<[3][2]> #SUP: 3',
 '<[3][3]> #SUP: 3',
 '<[4]> #SUP: 3',
 '<[4][2]> #SUP: 2',
 '<[4][3]> #SUP: 3',
 '<[4][3][2]> #SUP: 2',
 '<[5]> #SUP: 3',
 '<[5][1]> #SUP: 2',
 '<[5][1][2]> #SUP: 2',
 '<[5][1][3]> #SUP: 2',
 '<[5][1][3][2]> #SUP: 2',
 '<[5][2]> #SUP: 2',
 '<[5][2][3]> #SUP: 2',
 '<[5][3]> #SUP: 2',
 '<[5][3][2]> #SUP: 2',
 '<[5][6]> #SUP: 2',
 '<[5][6][2]> #SUP: 2',
```

```
 '<[5][6][3]> #SUP: 2',
 '<[5][6][3][2]> #SUP: 2',
 '<[6]> #SUP: 3',
 '<[6][2]> #SUP: 2',
 '<[6][2][3]> #SUP: 2',
 '<[6][3]> #SUP: 2',
 '<[6][3][2]> #SUP: 2']
```

Comparison with the bigger file KDD Cup

In [19]:
```
%time os.system("java -jar spmf.jar run PrefixSpan KDDCUP2000BMS

results=SPMFResultConverter("output.txt")
sorted(results)
```

```
CPU times: user 860 µs, sys: 2.05 ms, total: 2.91 ms
Wall time: 866 ms
```

Out[19]:
```
['<[10291]> #SUP: 472',
 '<[10295]> #SUP: 2009',
 '<[10295][10299]> #SUP: 323',
 '<[10295][10307]> #SUP: 916',
 '<[10295][10307][10311]> #SUP: 417',
 '<[10295][10307][10315]> #SUP: 335',
 '<[10295][10311]> #SUP: 738',
 '<[10295][10311][10315]> #SUP: 351',
 '<[10295][10315]> #SUP: 722',
 '<[10295][12695]> #SUP: 336',
 '<[10295][12895]> #SUP: 398',
 '<[10299]> #SUP: 721',
 '<[10299][10307]> #SUP: 460',
 '<[10303]> #SUP: 461',
 '<[10307]> #SUP: 2797',
 '<[10307][10311]> #SUP: 621',
 '<[10307][10315]> #SUP: 496'
```

Comparison of execution times between Python and Java implementations

Comparison of times between the two PrefixSpan implementations (Python and Java)

In [20]:

```
print ('Time for PrefixSpan Java: \n')
%time os.system("java -jar spmf.jar run PrefixSpan KDDCUP2000BMS
#with 0.34% PrefixSpan reports minsup = 203 sequences.

LogData=[]
#using url
LogData=SPMFFormatConverter('KDDCUP2000BMS1_spmf.txt')
print ('\nTime for PrefixSpan Python: \n')
%time result = prefixspan.prefixSpan (LogData, 203)
```

```
Time for PrefixSpan Java:

CPU times: user 1.87 ms, sys: 1.86 ms, total: 3.73 ms
Wall time: 833 ms

Time for PrefixSpan Python:

CPU times: user 37.7 s, sys: 413 ms, total: 38.1 s
Wall time: 40.2 s
```