SDSC3001 Tutorial 1

Frequent Pattern Mining

2024.09.12

About TAs

Tao HU

- PhD student under Prof. Zijun Zhang.
- TA of the course SDSC3001
- Email: taohu23-c@my.cityu.edu.hk

Lyuyi ZHU

- PhD student under Prof. Yu Yang.
- TA of the course SDSC3001
- Email: lyzhu9-c@my.cityu.edu.hk

Review of Frequent Pattern Mining (FPM)

Find all the frequent item sets that appear at least minimum support times

- Data Mining: extracting patterns from massive data
  - Pattern: a set of items, subsequences, or substructures that occur together frequently in a data set
- Motivation: uncovering inherent regularities in data

Frequently bought together

Total price: CDN$ 42.90

Add both to Cart

☑ **This item:** Huggies Natural Care Fragrance-Free Baby Wipes, Refill Pack, 1056 Count CDN$ 19.97 (CDN$ 0.02 / count)

☑ Playtex Diaper Genie Diaper Pail System Refills, 3 pack CDN$ 22.93 (CDN$ 7.64 / ring)

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- Frequent Pattern/Itemset Mining
  - Input: a transaction DB, *min_sup*
  - Output: all frequent itemsets

Review of Frequent Pattern Mining (FPM)

Algorithm 1 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

1: $L_1 \leftarrow \{frequent\ items\}$
2: **for** $k = 2$; $L_{k-1} \neq \emptyset$; $k \leftarrow k + 1$ **do**
3: $C_k \leftarrow$ candidates generated based on L_{k-1} **(Candidate Generation)**
4: Scan Transaction DB to count supports of itemsets in C_k **(Counting Supports)**
5: $L_k \leftarrow$ candidates in C_k with *min_sup*
6: **end for**
7: **return** $\cup_k L_k$

Review of Frequent Pattern Mining (FPM)

Algorithm 2 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

1: $L_1 \leftarrow \{frequent\ items\}$
2: **for** $k = 2$; $L_{k-1} \neq \emptyset$; $k \leftarrow k+1$ **do**
3: $\quad C_k \leftarrow$ candidates generated based on L_{k-1} (Candidate Generation)
4: $\quad$ Scan Transaction DB to count supports of itemsets in C_k (Counting Supports)
5: $\quad L_k \leftarrow$ candidates in C_k with *min_sup*
6: **end for**
7: **return** $\cup_k L_k$

Candidate Generation in Apriori

- C_k is generated based on L_{k-1}
  - Candidates should be extensions of itemsets in L_{k-1}
- Step 1: Self-joining L_{k-1}
  - Idea: use two $(k-1)$-itemsets in L_{k-1} to make a possibly frequent k-itemset
  - Every itemset is a string in alphabetical order (e.g. items are $a < b < ... < z$, $\{a, d, c, b\} = abcd$)
  - If $l_1[1 : k-2] = l_2[1 : k-2]$, and $l_1[k-1] < l_2[k-1]$, add $l_3 = l_1 \cup l_2$ to C_k (Prove the completeness by yourself)
- Step 2: Pruning candidates that are supersets of infrequent $(k-1)$-itemsets
  - The anti-monotonicity property of itemsets
  - Check every $(k-1)$-subset of a candidate

L: Frequent Pattern Set
C_k: Candidates

Input

min_sup=2

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1[st] scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

$C_k \leftarrow$ candidates generated based on L_{k-1}

1. Self-Joining: All possible combination $C_4^2 = 6$
2. Pruning candidates

Step 2: Pruning candidates that are supersets of infrequent $(k-1)$-itemsets

- The anti-monotonicity property of itemsets
- Check every $(k-1)$-subset of a candidate

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2[nd] scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3[rd] scan

L_3

Itemset	sup
{B, C, E}	2

Q: ? NO {A,B,C}{A,B,E}

A: {A,B} does not show up in L_2<- {A,B} Infrequent

- Any superset of an infrequent itemset must also be infrequent
  - {beer,diaper} is infrequent $\Rightarrow$ {beer,diaper,nuts} is infrequent
  - No superset of infrequent itemset should be generated
  - **Many item combinations can be pruned!**

Algorithm 2 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

```
L_1 ← {frequent items}
for k = 2; L_{k-1} ≠ ∅; k ← k + 1 do
    C_k ← candidates generated based on L_{k-1} (Candidate Generation)
    Scan Transaction DB to count supports of itemsets in C_k (Counting Supports)
    L_k ← candidates in C_k with min_sup
end for
return ∪_k L_k
```

Code implementation of FPM

- Jupyter Notebook
- Python
- C, Java, …

Download Data

https://archive.ics.uci.edu/dataset/352/online+retail

Online Retail

Donated on 11/5/2015

This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail.

Dataset Characteristics
Multivariate, Sequential, Time-Series

Subject Area
Business

Associated Tasks
Classification, Clustering

Attribute Type
Integer, Real

 # Instances
541909

 # Attributes
8

Information

Additional Information

This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail.The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers.

Data processing

First 10 rows of the dataset

1	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
2	536365	85123A	WHITE HANGING HEART	6	2010/12/1 8:26	2.55	17850	United Kingdom
3	536365	71053	WHITE METAL LANTERN	6	2010/12/1 8:26	3.39	17850	United Kingdom
4	536365	84406B	CREAM CUPID HEARTS (	8	2010/12/1 8:26	2.75	17850	United Kingdom
5	536365	84029G	KNITTED UNION FLAG I	6	2010/12/1 8:26	3.39	17850	United Kingdom
6	536365	84029E	RED WOOLLY HOTTIE WI	6	2010/12/1 8:26	3.39	17850	United Kingdom
7	536365	22752	SET 7 BABUSHKA NEST	2	2010/12/1 8:26	7.65	17850	United Kingdom
8	536365	21730	GLASS STAR FROSTED '	6	2010/12/1 8:26	4.25	17850	United Kingdom
9	536366	22633	HAND WARMER UNION J.	6	2010/12/1 8:28	1.85	17850	United Kingdom
10	536366	22632	HAND WARMER RED POL	6	2010/12/1 8:28	1.85	17850	United Kingdom

Data processing

1	InvoiceNo	StockCode
2	536365	85123A
3	536365	71053
4	536365	84406B
5	536365	84029G
6	536365	84029E
7	536365	22752
8	536365	21730
9	536366	22633
10	536366	22632

Itemset1:
{85123A, 71053, 84406B, 84029G, 84029E, 22752, 21730}

Itemset2:
{22633, 22632}

Data processing

In [11]:
```
import pandas as pd
import pickle

data = pd.read_excel("Online Retail.xlsx") # read excel by pandas
print(data)
data = data.values
data = data[:, 0:2] # we only need first two columns print(data)
order_list = list() # empty list, list of our final data
order_set = set() # empty orderSet
InvoiceNo = data[0, 0] # the current InvoiceNo
```

```
       InvoiceNo StockCode                          Description  Quantity  \
0         536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1         536365     71053                  WHITE METAL LANTERN         6
2         536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3         536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4         536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
...          ...       ...                                  ...       ...
541904    581587     22613          PACK OF 20 SPACEBOY NAPKINS        12
541905    581587     22899         CHILDREN'S APRON DOLLY GIRL          6
541906    581587     23254        CHILDRENS CUTLERY DOLLY GIRL          4
541907    581587     23255      CHILDRENS CUTLERY CIRCUS PARADE         4
541908    581587     22138        BAKING SET 9 PIECE RETROSPOT          3

               InvoiceDate  UnitPrice  CustomerID         Country
0      2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2      2010-12-01 08:26:00       2.75     17850.0  United Kingdom


[541909 rows x 8 columns]
```

Data processing

```
for i in range(len(data)):
    if data[i, 0] == InvoiceNo:
        # add a new item in orderSet
        order_set.add(str(data[i, 1]))
    else:
        # this orderSet is end
        order_list.append(order_set)
        InvoiceNo = data[i, 0]  # next InvoiceNo
        order_set = set()
        order_set.add(data[i, 1])
print(order_list[0:10])
```

```
[{'85123A', '22752', '84406B', '21730', '71053', '84029G', '84029E'}, {22633, '22632'}, {'21755', '22748', '22623', '22622', '22310', 84879, '22745', '21754', '21777', '84969', '48187', '22749'}, {22960, '22912', '22914', '22913'}, {21756}, {'22659', '21883', '21035', '21724', '22727', '21913', '21731', '22726', '21791', 'POST', '22544', '22492', 22728, '10002', '22631', '22900', '22629', '22661', '22540', '22326'}, {22086}, {22632, '22633'}, {'85123A', '82482', '21730', '22752', '37370', '82483', '21068', '21871', '84406B', '20679', '82486', '71053', '21071', '82494L', '84029G', '84029E'}, {21258}]
```

Review of Frequent Pattern Mining (FPM)

Algorithm 1 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

1: $\boxed{L_1 \leftarrow \{frequent\ \ items\}}$ → Initialization
2: **for** $k = 2$; $L_{k-1} \neq \emptyset$; $k \leftarrow k + 1$ **do**
3: $C_k \leftarrow$ candidates generated based on L_{k-1} **(Candidate Generation)**
4: Scan Transaction DB to count supports of itemsets in C_k **(Counting Supports)**
5: $L_k \leftarrow$ candidates in C_k with *min_sup*
6: **end for**
7: **return** $\cup_k L_k$

Apriori Algorithm

In [20]:
```
# the initial candidates_dict are all single item
candidates_dict = dict()
for order in order_list:
    for item in order:
        item = tuple([item])
        if item not in candidates_dict:
            candidates_dict[item] = 0    # add an item in candidates_dict with count 0

print("initial candidates_dict")
print(candidates_dict)
print("length of initial candidates_dict")
print(len(candidates_dict))
```

```
initial candidates_dict
{('85123A',): 0, ('22752',): 0, ('84406B',): 0, ('21730',): 0, ('71053',): 0, ('84029G',): 0, ('84029E',): 0, (22633,): 0, ('22632',): 0,
('21755',): 0, ('22748',): 0, ('22623',): 0, ('22622',): 0, ('22310',): 0, (84879,): 0, ('22745',): 0, ('21777',): 0, ('21754',): 0, ('481
87',): 0, ('22749',): 0, ('84969',): 0, (22960,): 0, ('22912',): 0, ('22914',): 0, ('22913',): 0, (21756,): 0, ('22659',): 0, ('21883',):
0, ('22326',): 0, ('21035',): 0, ('21724',): 0, ('22727',): 0, ('21913',): 0, ('21731',): 0, ('22726',): 0, ('21791',): 0, ('22544',): 0,
('22492',): 0, (22728,): 0, ('10002',): 0, ('22631',): 0, ('22900',): 0, ('22629',): 0, ('22661',): 0, ('22540',): 0, ('POST',): 0, (2208
6,): 0, (22632,): 0, ('22633',): 0, ('82482',): 0, ('37370',): 0, ('82483',): 0, ('21068',): 0, ('21871',): 0, ('20679',): 0, ('82486',):
0, ('21071',): 0, ('82494L',): 0, (21258,): 0, ('21733',): 0, (22114,): 0, ('84519A',): 0, ('20723',): 0, ('21977',): 0, ('85183B',): 0,
```

Review of Frequent Pattern Mining (FPM)

Algorithm 1 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

```
L_1 ← {frequent items}
for k = 2; L_{k-1} ≠ ∅; k ← k + 1 do
    C_k ← candidates generated based on L_{k-1} (Candidate Generation)
    Scan Transaction DB to count supports of itemsets in C_k (Counting Supports)
    L_k ← candidates in C_k with min_sup
end for
return ∪_k L_k
```

Use a function

Candidates Generation

In [19]:

```
def update_candidates_dict(frequent_list):
    old_candidates_list = list()
    for candidate in frequent_list:
        candidate = list(candidate)
        candidate.sort()
        old_candidates_list.append(candidate)

    # the size of old candidate
    size = len(old_candidates_list[0])

    new_candidates_list = list()
    # i = 0, j = 1, 2, 3, ..., m-1
    # i = 1, j = 2, ..., m-1
    # i = 2, j = 3, ..., m-1
    # ...
    # i = m-2, j = m-1
    for i in range(len(old_candidates_list) - 1):
        for j in range(i+1, len(old_candidates_list)):
            candidateA = list(old_candidates_list[i])
            candidateB = list(old_candidates_list[j])
            agree = True    # they have a possible father candidate
            for k in range(size-1):
                if candidateA[k] != candidateB[k]:
                    agree = False
                    break
            if agree:
                candidateC = candidateA.copy()
                candidateC.extend(candidateB)
                new_candidates_list.append(candidateC)

    candidates_dict = dict()
    for candidate in new_candidates_list:
        candidates_dict[tuple(candidate)] = 0    # add a candidate in candidates_dict with count 0
    return candidates_dict
```

K-1

l_1 (candidateA), l_2 (candidateB)

K-2

$l_1[1:k-2] = l_2[1:k-2]$,

K items

- C_k is generated based on L_{k-1}
  - Candidates should be extensions of itemsets in L_{k-1}
- Step 1: Self-joining L_{k-1}
  - Idea: use two $(k-1)$-itemsets in L_{k-1} to make a possibly frequent k-itemset
  - Every itemset is a string in alphabetical order (e.g. items are $a < b < ... < z$, $\{a, d, c, b\} = abcd$)
  - If $l_1[1:k-2] = l_2[1:k-2]$, and $l_1[k-1] < l_2[k-1]$, add $l_3 = l_1 \cup l_2$ to C_k (Prove the completeness by yourself)
- Step 2: Pruning candidates that are supersets of infrequent $(k-1)$-itemsets
  - The anti-monotonicity property of itemsets
  - Check every $(k-1)$-subset of a candidate

sort:

$l_1[k-1] < l_2[k-1]$,

- Every itemset is a string in alphabetical order (e.g. items are $a < b < ... < z$, $\{a, d, c, b\} = abcd$)

Candidates Generation

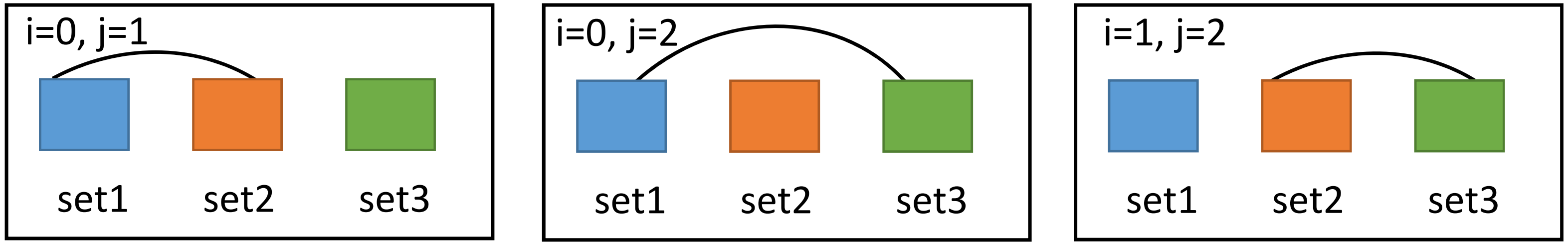
```python
old_candidates_list = list()
for candidate in frequent_list:
    candidate = list(candidate)
    candidate.sort()
    old_candidates_list.append(candidate)

# the size of old candidate
size = len(old_candidates_list[0])

new_candidates_list = list()
```

Candidates Generation

```python
for i in range(len(old_candidates_list) - 1):
    for j in range(i+1, len(old_candidates_list)):
        candidateA = list(old_candidates_list[i])
        candidateB = list(old_candidates_list[j])
        agree = True    # they have a possible father candidate
        for k in range(size-1):
            if candidateA[k] != candidateB[k]:
                agree = False
                break
        if agree:
            candidateC = candidateA.copy()
            candidateC.extend(candidateB)
            new_candidates_list.append(candidateC)
```



Candidates Generation: An Example

L4: [{abcd}, {abcg}, {abef}]

C5: ?

Here k=5, k-2=3

Candidates Generation: An Example

L4: [{abcd}, {abcg}, {abef}]

{<u>abc</u>d}, {<u>abc</u>g} -> “abc” = “abc”

{abcdg}

Candidates Generation: An Example

L4: [{abcd}, {abcg}, {abef}]

{<u>abc</u>d}, {<u>abe</u>f} -> "abc" $\neq$ "abe"

{<u>abc</u>g}, {<u>abe</u>f} -> "abc" $\neq$ "abe"

Candidates Generation: An Example

L4: [{abcd}, {abcg}, {abef}]

C5: [{abcdg}]

(Only 1 candidate)

Apriori Algorithm

```python
# core code
print()
while True:
    print()
    print("Start Scan the data set...")
    # go though the order_list
    for order in order_list:
        for candidate in candidates_dict:
            temp_set = set()
            for item in candidate:
                temp_set.add(item)
            if temp_set.issubset(order):      # check issubset
                candidates_dict[candidate] += 1   # count

    # check frequency
    frequent_list = list()
    for candidate in candidates_dict:
        if candidates_dict[candidate] >= min_sup:
            frequent_list.append(candidate)
    print("frequent_list:")
    print(frequent_list)
    print("length of frequent_list:")
    print(len(frequent_list))
    if len(frequent_list) == 0:
        break

    final_output_list.extend(frequent_list)

    # update candidates_dict
    candidates_dict = update_candidates_dict(frequent_list)
    print("lenght of new candidates_dict")
    print(len(candidates_dict))
```

Algorithm 1 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

1: $L_1 \leftarrow \{frequent\ items\}$
2: **for** $k = 2$; $L_{k-1} \neq \emptyset$; $k \leftarrow k+1$ **do**
3: $C_k \leftarrow$ candidates generated based on L_{k-1} **(Candidate Generation)**
4: Scan Transaction DB to count supports of itemsets in C_k **(Counting Supports)**
5: $L_k \leftarrow$ candidates in C_k with *min_sup*
6: **end for**
7: **return** $\cup_k L_k$

Apriori Algorithm

1-itemset

```
frequent_list:
[('85123A',), ('22752',), ('84406B',), ('71053',), ('84029G',), ('84029E',), ('22632',), ('21755',), ('22748',), ('22622',), ('22745',),
('21754',), ('48187',), ('22749',), ('22659',), ('22326',), ('21035',), ('22727',), ('21731',), ('22726',), ('21791',), ('22492',), ('2263
1',), ('22900',), ('22629',), ('22661',), ('POST',), ('22633',), ('82482',), ('82483',), ('21871',), ('20679',), ('82486',), ('82494L',),
('21733',), ('20723',), ('21977',), ('21094',), ('84991',), ('21033',), ('22352',), ('21975',), ('85099C',), ('84997C',), ('84997B',), ('2
1931',), ('20725',), ('21212',), ('21929',), ('21559',), ('82567',), ('22646',), ('22411',), ('15056BL',), ('21175',), ('22083',), ('2208
6',), ('21523',), ('15056N',), ('22771',), ('21934',), ('21672',), ('21533',), ('22637',), ('22644',), ('21169',), ('84832',), ('21166',),
('22379',), ('22381',), ('22798',), ('21912',), ('22464',), ('22469',), ('22457',), ('22424',), ('22189',), ('22427',), ('21340',), ('2247
0',), ('84755',), ('22960',), ('22662',), ('22961',), ('85049A',), ('22663',), ('85099B',), ('22969',), ('22192',), ('85014B',), ('2219
3',), ('22195',), ('22191',), ('85014A',), ('22196',), ('22910',), ('22654',), ('22963',), ('21485',), ('22197',), ('21086',), ('21080',),
('84030E',), ('22962',), ('84970S',), ('22174',), ('22553',), ('21980',), ('21484',), ('22557',), ('21891',), ('84879',), ('21889',), ('22
502',), ('22619',), ('22865',), ('22652',), ('22558',), ('85152',), ('22866',), ('22729',), ('22730',), ('22867',), ('22114',), ('2131
4',), ('21479',), ('22112',), ('21232',), ('48185',), ('22835',), ('22111',), ('20726',), ('22766',), ('22384',), ('22382',), ('82581',),
('22467',), ('22549',), ('82580',), ('22851',), ('22435',), ('85049E',), ('21122',), ('85150',), ('82578',), ('22413',), ('22804',), ('219
07',), ('20992',), ('22768',), ('21328',), ('20749',), ('21213',), ('22296',), ('20728',), ('84380',), ('22383',), ('84992',), ('22417',),

length of frequent_list:
590
```

2-itemset

```
frequent_list:
[('85123A', '82482'), ('85123A', '82494L'), ('85123A', '21733'), ('85123A', '20725'), ('85123A', '21212'), ('85123A', '22469'), ('85123A',
'22457'), ('85123A', '22470'), ('85123A', '85099B'), ('85123A', '22197'), ('85123A', '84879'), ('85123A', '22384'), ('85123A', '22804'),
('85123A', '22383'), ('85123A', '20727'), ('85123A', '22178'), ('85123A', '22423'), ('85123A', '22666'), ('85123A', '47566'), ('85123A',
'22720'), ('21755', '21754'), ('22748', '22745'), ('22748', '22746'), ('22745', '22746'), ('22659', '22629'), ('22659', '22630'), ('2232
6', '22328'), ('22727', '22726'), ('22727', '22729'), ('22727', '22730'), ('22727', '22423'), ('22727', '22728'), ('22727', '22725'), ('21
731', '85099B'), ('21731', 'DOT'), ('22726', '22729'), ('22726', '22730'), ('22726', '22728'), ('22726', '22725'), ('21791', '21790'), ('2
2629', '85099B'), ('22629', '22630'), ('82482', '82494L'), ('82482', '85099B'), ('82483', '82486'), ('20723', '20725'), ('20723', '2121
2'), ('20723', '85099B'), ('20723', '20728'), ('20723', '20727'), ('20723', '22355'), ('20723', '22356'), ('20723', '20724'), ('20723', '2
0719'), ('21977', '84991'), ('21977', '21212'), ('21094', '21086'), ('21094', '21080'), ('84991', '21212'), ('84991', '84992'), ('21975',
'21212'), ('85099C', '21931'), ('85099C', '20725'), ('85099C', '21929'), ('85099C', '22411'), ('85099C', '85099B'), ('85099C', '20713'),
('85099C', '85099F'), ('85099C', '22386'), ('85099C', '21930'), ('85099C', '20712'), ('85099C', '21928'), ('85099C', 'DOT'), ('85099C', '2
3199'), ('21931', '20725'), ('21931', '21929'), ('21931', '22411'), ('21931', '22379'), ('21931', '85099B'), ('21931', '22197'), ('21931',
'22383'), ('21931', '20727'), ('21931', '20713'), ('21931', '85099F'), ('21931', '22386'), ('21931', '21930'), ('21931', '20712'), ('2193
1', '22355'), ('21931', '20711'), ('21931', '21928'), ('21931', 'DOT'), ('21931', '20724'), ('21931', '22385'), ('21931', '23201'), ('2193
1', '23199'), ('21931', '23202'), ('21931', '23203'), ('20725', '21212'), ('20725', '22411'), ('20725', '22662'), ('20725', '85099B'), ('2

length of frequent_list:
408
```

8-itemset

```
frequent_list:
[('21931', '21931', '22386', '22411', '21931', '21931', '22386', '85099B')]
length of frequent_list:
1
```

Thank you!