```
from itertools import chain, combinations,islice
from collections import defaultdict
from time import time
import pandas as pd
import operator


def run_apriori(infile, min_support, min_conf):
    """
    Run the Apriori algorithm. infile is a record iterator.
    Return:
        rtn_items: list of (set, support)
        rtn_rules: list of ((preset, postset), confidence)
    """
    one_cand_set, all_transactions = gen_one_item_cand_set(infile)

    set_count_map = defaultdict(int)  # maintains the count for each set

    one_freq_set, set_count_map = get_items_with_min_support(
        one_cand_set, all_transactions, min_support, set_count_map)

    freq_map, set_count_map = run_apriori_loops(
        one_freq_set, set_count_map, all_transactions, min_support)

    rtn_items = get_frequent_items(set_count_map, freq_map)
    rtn_rules = get_frequent_rules(set_count_map, freq_map, min_conf)

    return rtn_items, rtn_rules


def gen_one_item_cand_set(input_fileator):
    """
    Generate the 1-item candidate sets and a list of all the transactions.
    """
    all_transactions = list()
    one_cand_set = set()
    for record in input_fileator:
        transaction = frozenset(record)
        all_transactions.append(transaction)
        #========================#
        # STRART YOUR CODE HERE  #
        #========================#
        for item in transaction:
            new_set = set()
            new_set.add(item)
            if frozenset(new_set) not in one_cand_set:
                one_cand_set.add(frozenset(new_set))
        #========================#
        #   END YOUR CODE HERE   #
        #========================#
    return one_cand_set, all_transactions


def get_items_with_min_support(item_set, all_transactions, min_support,
                               set_count_map):
    """
    item_set is a set of candidate sets.
    Return a subset of the item_set
    whose elements satisfy the minimum support.
```

```python
    """
    rtn = set()
    local_set = defaultdict(int)

    for item in item_set:
        for transaction in all_transactions:
            if item.issubset(transaction):
                set_count_map[item] += 1
                local_set[item] += 1

    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    for item, count in local_set.items():
        if local_set[item] >= min_support:
            rtn.add(item)
    #========================#
    #   END YOUR CODE HERE   #
    #========================#



    return rtn, set_count_map


def run_apriori_loops(one_cand_set, set_count_map, all_transactions,
                      min_support):
    """
    Return:
        freq_map: a dict
            {<length_of_set_l>: <set_of_frequent_itemsets_of_length_l>}
        set_count_map: updated set_count_map
    """
    freq_map = dict()
    current_l_set = one_cand_set
    i = 1
    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    while (current_l_set != set([])):
        freq_map[i] = current_l_set
        current_l_set = join_set(current_l_set, i+1)
        current_c_set, set_count_map = get_items_with_min_support(current_l_set, all_transactions, min_support, set_count_map)
        current_l_set = current_c_set

        i += 1
    #========================#
    #   END YOUR CODE HERE   #
    #========================#

    return freq_map, set_count_map


def get_frequent_items(set_count_map, freq_map):
    """ Return frequent items as a list. """
    rtn_items = []
    for key, value in freq_map.items():
            [(tuple(item), get_support(set_count_map, item))
             for item in value])
    return rtn_items


def get_frequent_rules(set_count_map, freq_map, min_conf):
    """ Return frequent rules as a list. """
    rtn_rules = []
    for key, value in islice(freq_map.items(),1,None):
        for item in value:
            _subsets = map(frozenset, [x for x in subsets(item)])
            for element in _subsets:
                remain = item.difference(element)
                if len(remain) > 0:
               #========================#
               # STRART YOUR CODE HERE  #
               #========================#
                    confidence = float(set_count_map[element.union(remain)]) / float(set_count_map[element])
               #========================#
               #   END YOUR CODE HERE   #
               #========================#
                    if confidence >= min_conf:
                        rtn_rules.append(
                            ((tuple(element), tuple(remain)), confidence))
    return rtn_rules


def get_support(set_count_map, item):
    """ Return the support of an item. """
    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    sup_item = set_count_map[item]
    #========================#
    #   END YOUR CODE HERE   #
    #========================#
    return sup_item


def join_set(s, l):
    """
    Join a set with itself .
    Return a set whose elements are unions of sets in s with length==l.
    """
    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    join_set = set()
    for set_one in s:
        for set_two in s:
            if set_one is not set_two:
                joint_set = set_one.union(set_two)

                if len(joint_set) == l:
                    join_set.add(joint_set)
    #========================#
    #   END YOUR CODE HERE   #
    #========================#
```

```python


def subsets(x):
    """ Return non =-empty subsets of x. """
    return chain(*[combinations(x, i + 1) for i, a in enumerate(x)])


def print_items_rules(items, rules, ignore_one_item_set=False, name_map=None):
    for item, support in sorted(items, key=operator.itemgetter(1)):
        if len(item) == 1 and ignore_one_item_set:
            continue
        print ('item: ')
        print (convert_item_to_name(item, name_map), support)
    print ('\n------------------------ RULES:')
    for rule, confidence in sorted(
            rules, key=operator.itemgetter(1)):
        pre, post = rule
        print ('Rule: ')
        print( convert_item_to_name(pre, name_map), convert_item_to_name(post, name_map),confidence)


def convert_item_to_name(item, name_map):
    """ Return the string representation of the item. """
    if name_map:
        return ','.join([name_map[x] for x in item])
    else:
        return str(item)


def read_data(fname):
    """ Read from the file and yield a generator. """
    file_iter = open(fname, 'rU')
    for line in file_iter:
        line = line.strip().rstrip(',')
        record = frozenset(line.split(','))
        yield record


def read_name_map(fname):
    """ Read from the file and return a dict mapping ids to names. """
    df = pd.read_csv(fname, sep=',\t ', header=None, names=['id', 'name'],
                     engine='python')
    return df.set_index('id')['name'].to_dict()



```

CS145 Howework 5

Important Note: HW4 is due on **11:59 PM PT, Dec 4 (Friday, Week 9)**. Please submit through GradeScope.

Print Out Your Name and UID

Name: Ali Mirabzadeh, UID: 305179067

Before You Start

You need to first create HW5 conda environment by the given `cs145hw5.yml` file, which provides the name and necessary packages for this tasks. If you have `conda` properly installed, you may create, activate or deactivate by the following commands:

```
conda env create -f cs145hw5.yml
conda activate hw4
conda deactivate
```

OR

```
conda env create --name NAMEOFYOURCHOICE -f cs145hw5.yml
conda activate NAMEOFYOURCHOICE
conda deactivate
```

To view the list of your environments, use the following command:

```
conda env list
```

More useful information about managing environments can be found here (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html).

You may also quickly review the usage of basic Python and Numpy package, if needed in coding for matrix operations.

In this notebook, you must not delete any code cells in this notebook. If you change any code outside the blocks (such as some important hyperparameters) that you are allowed to edit (between STRART/END YOUR CODE HERE), you need to highlight these changes. You may add some additional cells to help explain your results and observations.

In [1]:

```
import numpy as np
import pandas as pd
import sys
import random
import math
import matplotlib.pyplot as plt
from graphviz import Digraph
from IPython.display import Image
from scipy.stats import multivariate_normal
%load_ext autoreload
%autoreload 2
```

If you can successfully run the code above, there will be no problem for environment setting.

1. Frequent Pattern Mining for Set Data (25 pts)

Table 1

TID	Items
1	b,c,j
2	a,b,d
3	a,c
4	b,d
5	a,b,c,e
6	b,c,k
7	a,c
8	a,b,e,i
9	b,d
10	a,b,c,d

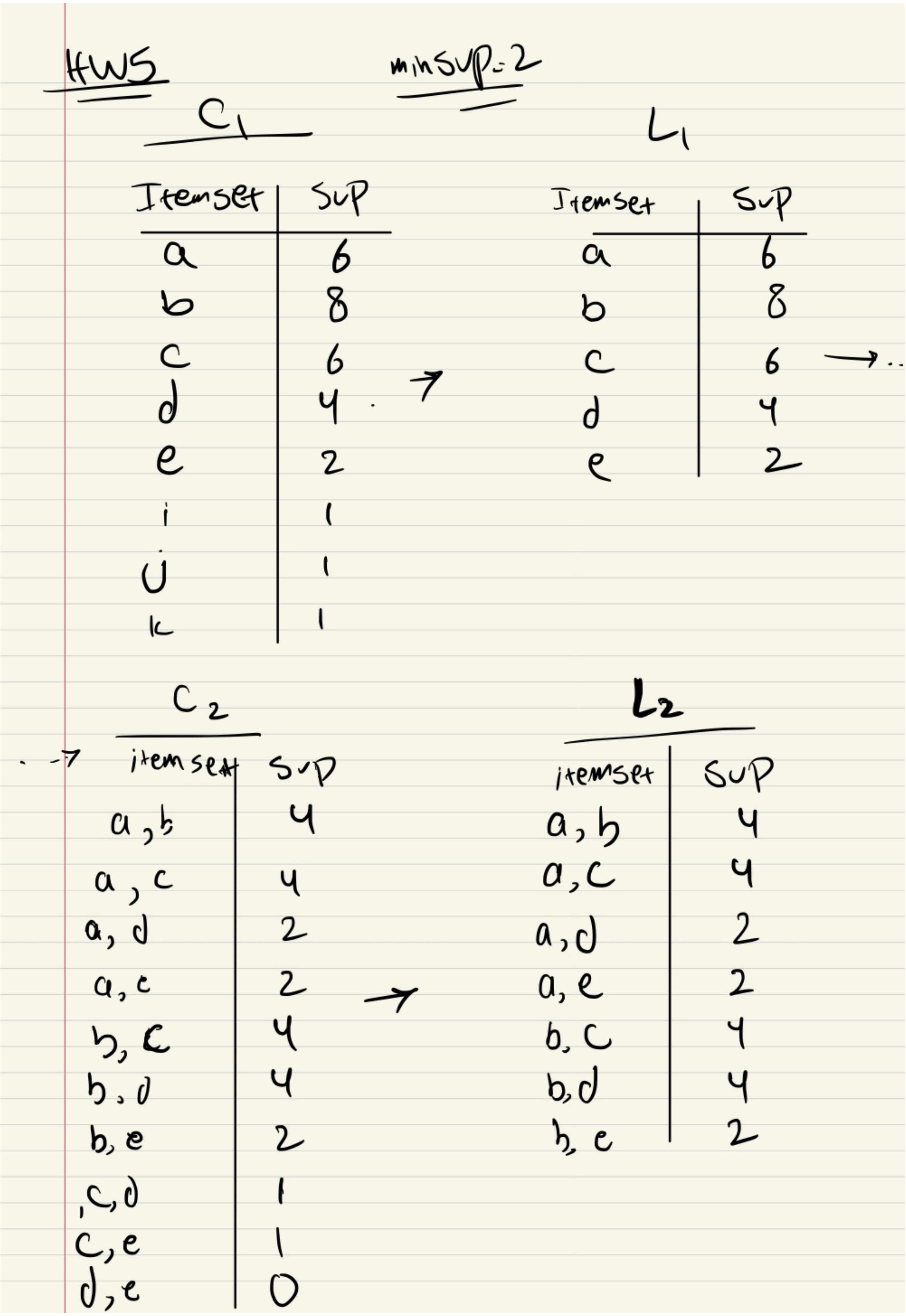
Given a transaction database shown in Table 1, answer the following questions. Let the parameter `min_support` be 2.

Questions

1.1 Apriori Algorothm (16 pts) .

Note: This is a "question-answer" style problem. You do not need to code anything and you are required to calculate by hand (with a scientific calculator). Find all the frequent patterns using Apriori Algorithm.
a. C_1
b. L_1
c. C_2
d. L_2

e. C_3
f. L_3
g. C_4
h. L_4



HW5 minSup=2

C1

Itemset	Sup
a	6
b	8
c	6
d	4
e	2
i	1
j	1
k	1

→

L1

Itemset	Sup
a	6
b	8
c	6
d	4
e	2

→ ..

C2

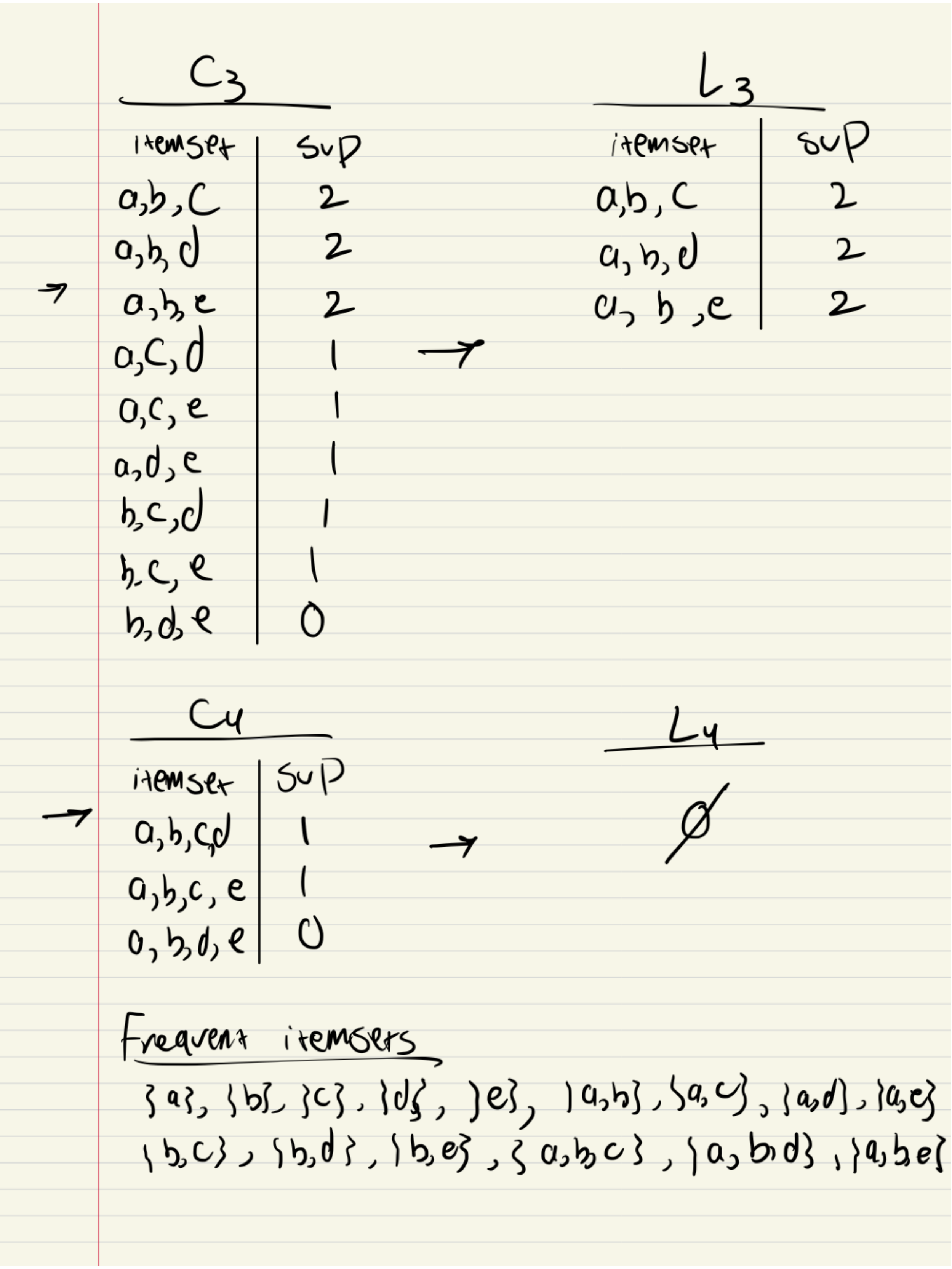
.. →

itemset	sup
a, b	4
a, c	4
a, d	2
a, e	2
b, c	4
b, d	4
b, e	2
c, d	1
c, e	1
d, e	0

→

L2

itemset	sup
a, b	4
a, c	4
a, d	2
a, e	2
b, c	4
b, d	4
b, e	2



C3

itemset	sup
a,b,c	2
a,b,d	2
a,b,e	2
a,c,d	1
a,c,e	1
a,d,e	1
b,c,d	1
b,c,e	1
b,d,e	0

L3

itemset	sup
a,b,c	2
a,b,d	2
a,b,e	2

C4

itemset	sup
a,b,c,d	1
a,b,c,e	1
a,b,d,e	0

L4

∅

Frequent itemsets

{a}, {b}, {c}, {d}, {e}, {a,b}, {a,c}, {a,d}, {a,e}
{b,c}, {b,d}, {b,e}, {a,b,c}, {a,b,d}, {a,b,e}

1.2 FP-tree (9 pts)

(a)Construct the FP-tree of the table.
(b) For the item d, show its conditional pattern base (projected database) and conditional FP-tree
You may use Package `graphviz` to generate graph

(https://graphviz.readthedocs.io/en/stable/manual.html (https://graphviz.readthedocs.io/en/stable/manual.html)) (Bonus point: 5pts) or draw by hand.

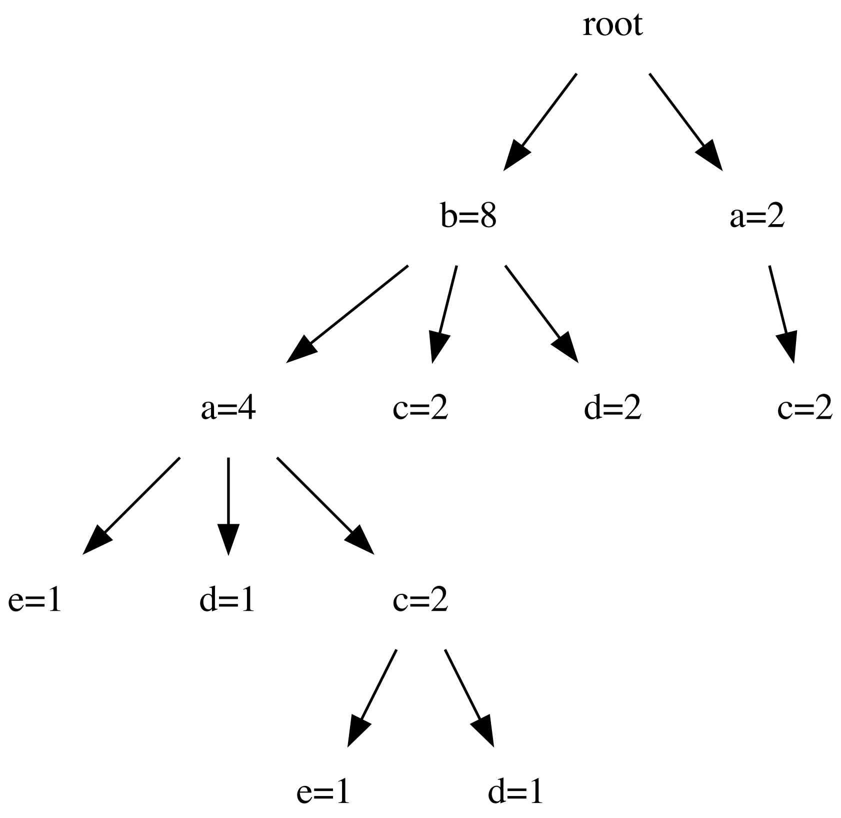
In [29]:
```
ps = Digraph(name='pet-shop', node_attr={'shape': 'plaintext'})
ps.node('R', 'root')
ps.node('A', 'b=8')
ps.node('B', 'a=2')
ps.edge('B', 'c=2')
ps.edges(['RB', 'RA'])

ps.node('C', 'a=4')
ps.node('D', 'c=2')
ps.node('E', 'd=2')
ps.edges(['AC', 'AD', 'AE'])

ps.node('F', 'e=1')
ps.node('G', 'd=1')
ps.node('H', 'c=2')
ps.edges(['CF', 'CG', 'CH'])

ps.node('I', 'e=1')
ps.node('J', 'd=1')
ps.edges(['HI', 'HJ'])
ps
```

Out[29]:



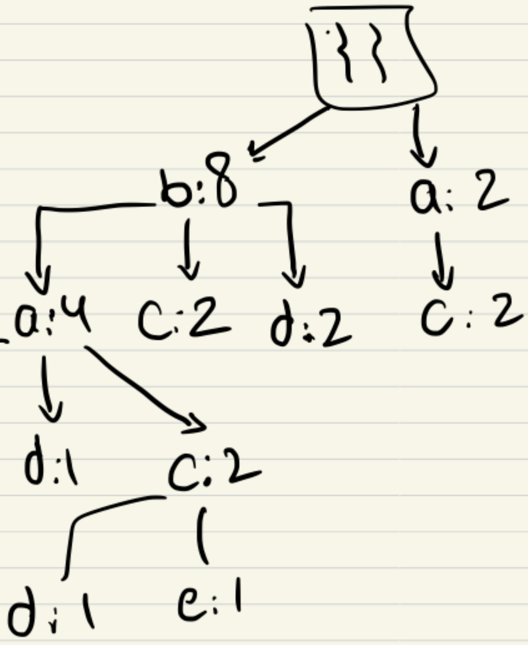
(c) Find frequent patterns based on d's conditional FP-tree

1.2

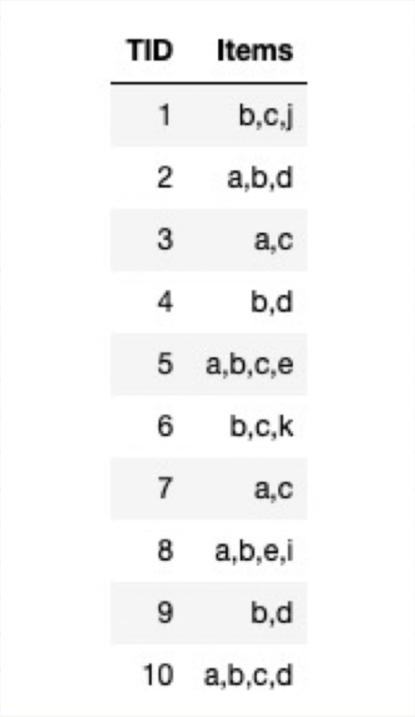
a,

order

F_list	
b	8
a	6
c	6
d	4
e	2



Ordered frequent items

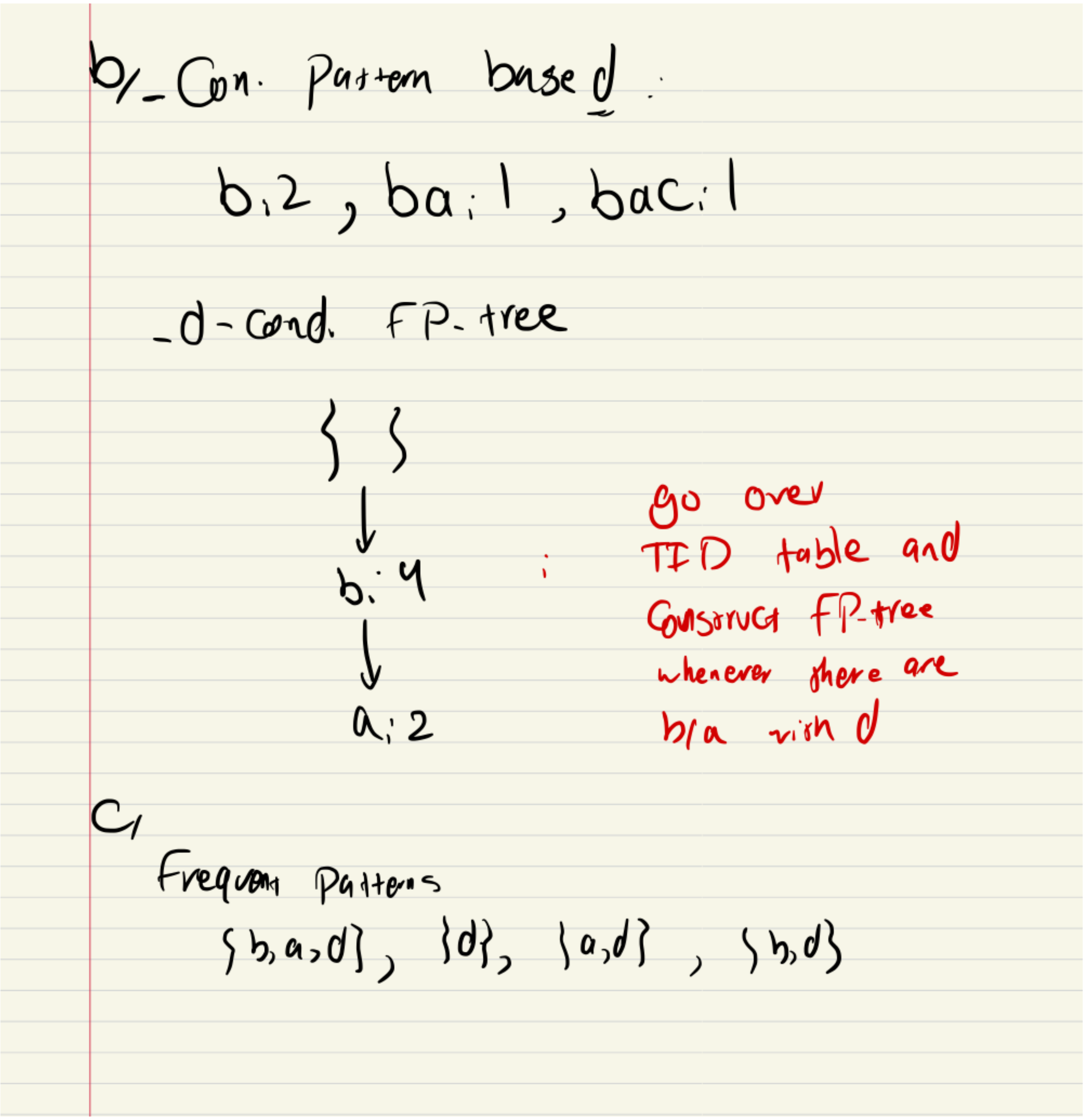


TID	Items
1	b,c,j
2	a,b,d
3	a,c
4	b,d
5	a,b,c,e
6	b,c,k
7	a,c
8	a,b,e,i
9	b,d
10	a,b,c,d

frequent itemes

order

b,C
a, b, d
a,C
b,d
a,b,c,e
b,C
a, c
a,b,e
b,d
a,b,c,d



b/_ Con. Pattern base d :

b:2 , ba:1 , bac:1

_d - Cond. FP- tree

{ }

↓

b: 4 ; go over TID table and Construct FP-tree whenever there are b/a with d

↓

a: 2

c/

Frequent Patterns

{b,a,d}, {d}, {a,d} , {b,d}

2. Apriori for Yelp (50 pts)

In `apriori.py` , fill the missing lines. The parameters are set as `min_suppport=50` and `min_conf` = 0.25, and `ignore_one_iter_set=True` . Use the Yelp data `yelp.csv` and `id_nams.csv` , and run the following cell and report the frequent patterns and rules associated with it.

In [3]:

```
#No need to modify
from hw5code.apriori import *
input_file = read_data('./data/yelp.csv')
min_support = 50
min_conf = 0.25
items, rules = run_apriori(input_file, min_support, min_conf)
name_map = read_name_map('./data/id_name.csv')
print_items_rules(items, rules, ignore_one_item_set=True, name_map=name_map
```

```
item:
"Holsteins Shakes & Buns","Wicked Spoon" 51
item:
"Secret Pizza","Wicked Spoon" 52
item:
"Earl of Sandwich","Wicked Spoon" 52
item:
"Wicked Spoon","The Cosmopolitan of Las Vegas" 54
item:
"Wicked Spoon","Mon Ami Gabi" 57
item:
"Bacchanal Buffet","Wicked Spoon" 63

------------------------ RULES:
Rule:
"Secret Pizza" "Wicked Spoon" 0.2561576354679803
Rule:
"The Cosmopolitan of Las Vegas" "Wicked Spoon" 0.27692307692307694
Rule:
"Holsteins Shakes & Buns" "Wicked Spoon" 0.3148148148148148
```

What do these results mean? Do a quick Google search and briefly interpret the patterns and rules mined from Yelp in 50 words or less.

Seems these are Las Vegas locations like Wicked Spoon is a biffet there and we can see here bases on apriori, it has a high frequency. Also, we can see from the RULES that Yelping other locations, there is a degree of confidence that they Yelped Wicked Spoon as well

3. Correlation Analysis (10 pts)

Note: This is a "question-answer" style problem. You do not need to code anything and you are required to calculate by hand (with a scientific calculator).

Table 2

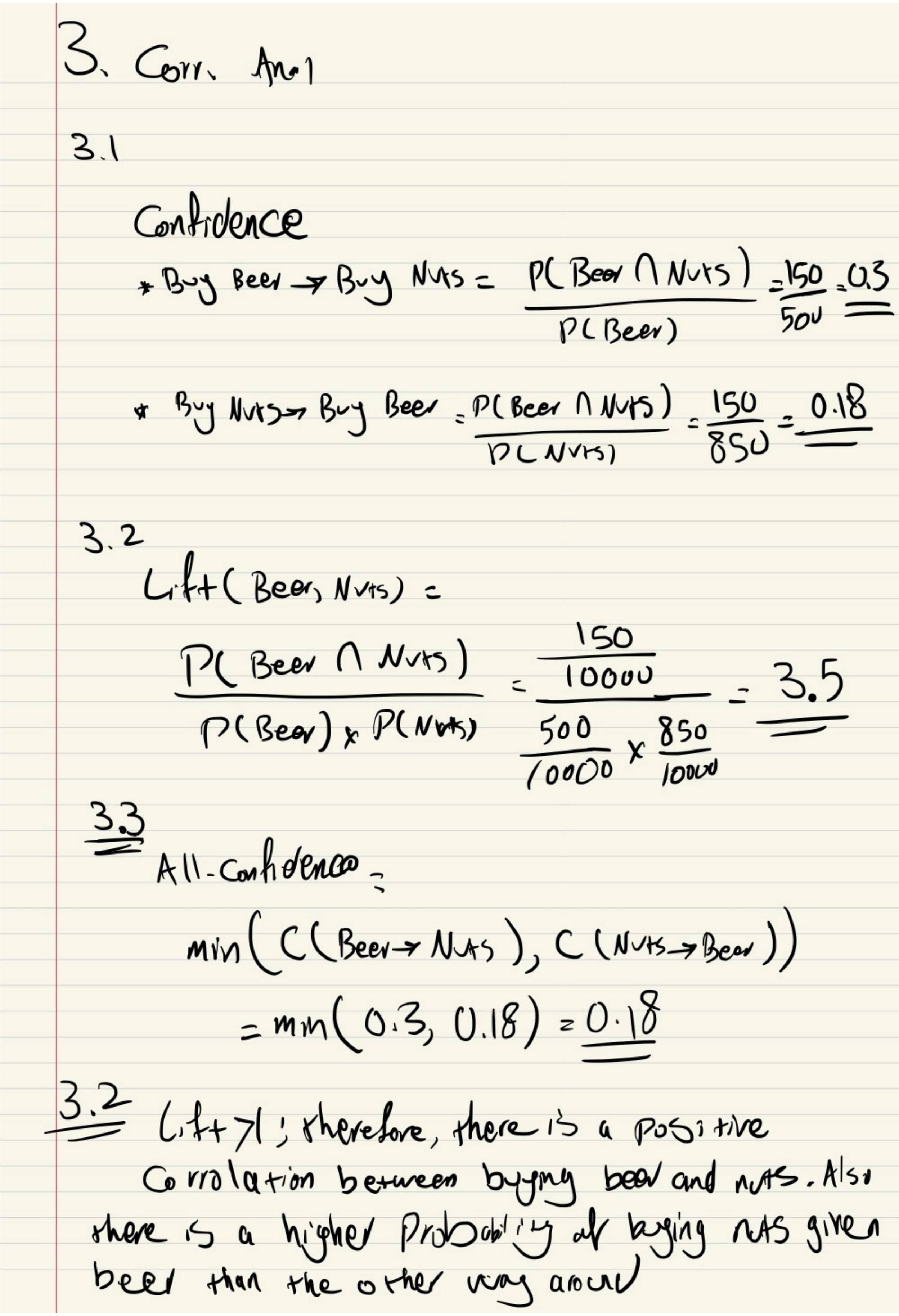
---	Beer	No Beer	Total
Nuts	150	700	850
No Nuts	350	8800	9150

---	Beer	No Beer	Total
Total	500	9500	10000

Table 2 shows how many transactions containing beer and/or nuts among 10000 transactions.

Answer the following questions:

3.1 Calculate `confidence` , `lift` and `all_confidence` between buying beer and buying nuts.
3.2 What are you conclusions of the relationship between buying beer and buying nuts? Justify your conclusion with the previous measurements you calculated in 3.1.



3. Corr. Anal

3.1

Confidence

$$* \text{ Buy Beer} \rightarrow \text{Buy Nuts} = \frac{P(\text{Beer} \cap \text{Nuts})}{P(\text{Beer})} = \frac{150}{500} = \underline{\underline{0.3}}$$

$$* \text{ Buy Nuts} \rightarrow \text{Buy Beer} = \frac{P(\text{Beer} \cap \text{Nuts})}{P(\text{Nuts})} = \frac{150}{850} = \underline{\underline{0.18}}$$

3.2

$$\text{Lift}(\text{Beer}, \text{Nuts}) = \frac{P(\text{Beer} \cap \text{Nuts})}{P(\text{Beer}) \times P(\text{Nuts})} = \frac{\frac{150}{10000}}{\frac{500}{10000} \times \frac{850}{10000}} = \underline{\underline{3.5}}$$

3.3

$$\text{All-confidence} = \min(C(\text{Beer} \rightarrow \text{Nuts}), C(\text{Nuts} \rightarrow \text{Beer})) = \min(0.3, 0.18) = \underline{\underline{0.18}}$$

3.2 Lift>1; therefore, there is a positive Corrolation between buying beer and nuts. Also there is a higher Probability of buying nuts given beer than the other way around

4. Sequential Pattern Mining (GSP Algorithm) (15 pts)

Note: This is a "question-answer" style problem. You do not need to code anything and you are required to calculate by hand (with a scientific calculator).

4.1 For a sequence $s =< ab(cd)(ef) >$, how many events or elements does it contain? What is the length of s? How many non-empty subsequences does s contain?
4.2 Suppose we have $L_3 = \{< (ac)e >, < b(cd) >, < bce >, < a(cd) >, < (ab)d >, < (ab)c >\}$, as the requent 3-sequences, write down all the candidate 4-sequences C_4 with the details of the join and pruning steps.

4.1. It contains 4 elements with a length of 6. For Subsequence: 2^6 -1 = 64 -1 =63 combinations
4.2. Join:
1.$< b(cd) >$ and $< (ab)c >$ to form $< (ab)(cd) >$
2. $< bce >$ and $< (ab)c >< bce >$ to form $< (ab)ce >$
Prune: check if all length-3 subsequence of above results in L3
prune $< (ab)ce >$ as $< (ab)e >$ can't be found in L3
L4: $< (ab)(cd) >$

5 Bonus Question (10 pts)

1.In FP-tree, what will happen if we use `ascending` instead `descending` in header table?
2.Describe CloSpan (`Mining closed sequential patterns: CloSpan (Yan, Han & Afshar @SDM’03)`). Compare with algorithms we discussed in class.

1. In asneding, in creating of FP-tree will make more branches as the more freqent itemsets appear towards the end
2. in CloSpan instead of mining the complete set of frequent subsequences, we mine frequent closed subsequences. That's why this algorrithm is more efficient than the one discussed in class. Also it's really good for long sequence of data

End of Homework 5 :)

After you've finished the homework, please print out the entire `ipynb` notebook and four `py` files into one PDF file. Make sure you include the output of code cells and answers for questions. Prepare submit it to GradeScope. Also this time remember assign the pages to the questions on GradeScope