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: Danning Yu, UID: 305087992

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.

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 [2]:
```
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

Note: the format is given as {pattern}:frequency.

a. {a}:6, {b}:8, {c}:6, {d}:4, {e}:2, {i}:1, {j}:1, {k}:1
b. {a}:6, {b}:8, {c}:6, {d}:4, {e}:2
c. {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
d. {a, b}:4, {a, c}:4, {a, d}:2, {a, e}:2, {b, c}:4, {b, d}:4, {b, e}:2

e. {a, b, c}:2, {a, b, d}:2, {a, b, e}:2. (Pruned due to containing an infrequent subset: {a, c, d}, {a, c, e}, {a, d, e}, {b, c, d}, {b, c, e}, {b, d, e})
f. {a, b, c}:2, {a, b, d}:2, {a, b, e}:2
g. None (Pruned due to containing an infrequent subset: {a, b, c, d}, {a, b, c, e}, {a, b, d, e})
h. None

Thus, all frequent patterns are {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) (Bonus point: 5pts) or draw by hand.

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

a. Frequent 1-itemsets in sorted order (sort by frequency, use alphabetical order to break ties): {b}:8, {a}:6, {c}:6, {d}:4, {e}:2

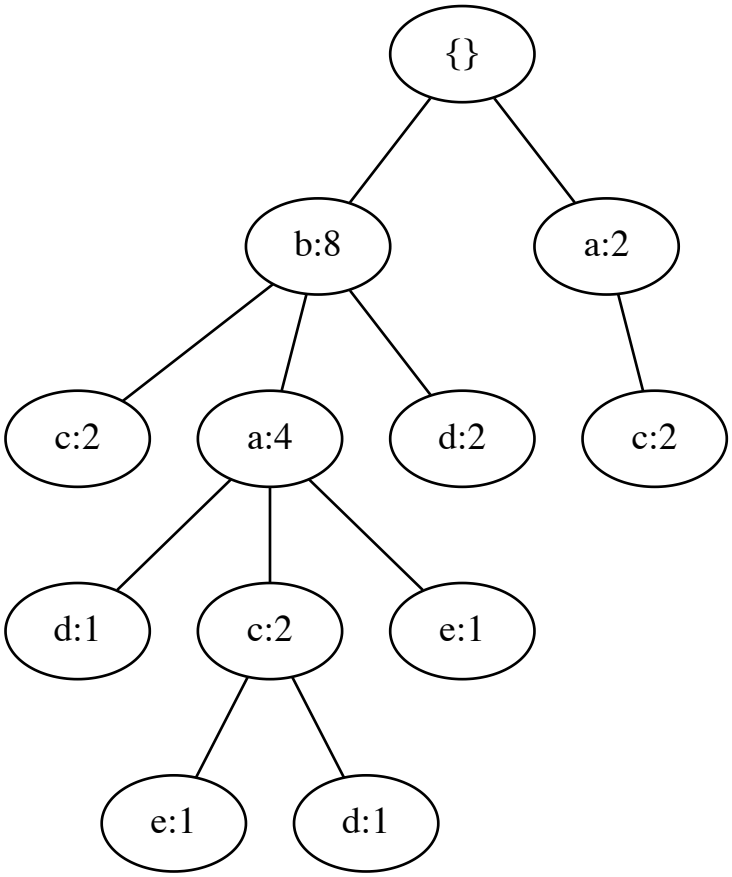
TID	Ordered Frequent Items
1	{b, c}
2	{b, a, d}
3	{a, c}
4	{b, d}
5	{b, a, c, e}
6	{b, c}
7	{a, c}
8	{b, a, e}
9	{b, d}
10	{b, a, c, d}

From this, the following graph was generated. See below the graph for parts (b) and (c).

In [22]:
```
from graphviz import Graph
dot = Graph(comment="Question 1.2a: FP Tree", format="png")
dot.node('root', '{}')
dot.node('1b8', 'b:8')
dot.node('1a2', 'a:2')
dot.edge('root', '1b8')
dot.edge('root', '1a2')

dot.node('2c2', 'c:2')
dot.edge('1b8', '2c2')

dot.node('2a4', 'a:4')
```

```
dot.edge('1b8', '2a4')

dot.node('2d2', 'd:2')
dot.edge('1b8', '2d2')

dot.node('3d1', 'd:1')
dot.edge('2a4', '3d1')

dot.node('3c2', 'c:2')
dot.edge('2a4', '3c2')

dot.node('3e1', 'e:1')
dot.edge('2a4', '3e1')

dot.node('4e1', 'e:1')
dot.edge('3c2', '4e1')

dot.node('4d1', 'd:1')
dot.edge('3c2', '4d1')

dot.node('2c2-1', 'c:2')
dot.edge('1a2', '2c2-1')

dot
```

Out[22]:



b. {b, a}:1, {b, a, c}:1, {b}:2

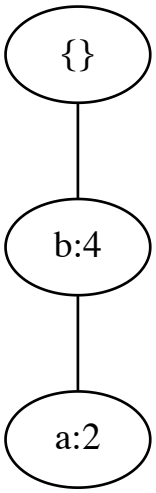
The conditional FP tree on `d` is given below. We remove the element "c" because it does not meet the `min_support` threshold of 2. The answer to (c) comes after the conditional FP tree.

In [48]:

```
from graphviz import Graph
dot2 = Graph(comment="Question 1.2b: FP Tree", format="png")
```

```
dot2.node('root', '{}')
dot2.node('1b4', 'b:4')
dot2.node('2a2', 'a:2')
dot2.edge('root', '1b4')
dot2.edge('1b4', '2a2')

dot2
```

Out[48]:



c. Frequent patterns based on `d` : {d}, {b, d}, {a, b, d}, {a, 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 [22]:

```
#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:
"Wicked Spoon","Holsteins Shakes & Buns" 51
item:
"Wicked Spoon","Secret Pizza" 52
item:
"Wicked Spoon","Earl of Sandwich" 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.

These are restaurants in the Las Vegas strip. Each frequent pattern indicates restaurants appearing together in the dataset; for example, Wicked Spoon and Secret Pizza were together 52 times. The rules say how likely that a transaction with Wicked Spoon also includes Secret Pizza; in this case, 25.6% chance.

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
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.

Question 3.1

Confidence(buying nuts => buying beer) = $P(\text{buying beer}|\text{buying nuts}) = \frac{150}{850} = 0.1765$
Confidence(buying beer => buying nuts) = $P(\text{buying nuts}|\text{buying beer}) = \frac{150}{500} = 0.3000$

Lift = $\frac{\text{support}(A \cup B)}{\text{support}(A)\text{support}(B)} = \frac{\frac{150}{10000}}{\left(\frac{500}{10000}\right)\left(\frac{850}{10000}\right)} = 3.5294$

All_confidence = $\min\left(P(\text{buying beer}|\text{buying nuts}), P(\text{buying nuts}|\text{buying beer})\right)$
All_confidence = $\min(0.1765, 0.3000) = 0.1765$

Question 3.2

Based on the value of `lift` , which is 3.5294, this indicates a positive correlation between buying beer and buying nuts. However, looking at `confidence` and `all_confidence` , they have low values, so while there is a positive correlation, it's not very strong. The value of lift is skewed by the large number of null-transactions and so is not a good measure of correlation. Finally, based on the calculations, the chance of buying nuts given that beer was bought (0.3) is slightly higher than the chance of buying beer given that nuts were bought (0.1765).

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.

Question 4.1
There are 4 elements and the length of s is 6.

To count the number of nonempty subsequences, we can include/exclude a or b. For the group (cd), we can include c or d, both, or neither, so there are 4 ways. The same is true for (ef). Finally, we can't have an empty subsequence, so subtract 1 from the result. The number of non-empty subsequences is $2(2)(4)(4) - 1 = 63$.

Question 4.2
The joins that are possible:
$< (ab)c > and < b(cd) >$ gives $< (ab)(cd) >$
$< (ab)c > and < bce >$ gives $< (ab)ce >$

Then we check all possible subsequences that are 3-sequences:
For $< (ab)(cd) >$, the subsequences are $< (ab)c >, < a(cd) >, < (ab)d >, < b(cd) >$. All are frequent 3-subsequences, so none are pruned.
For $< (ab)ce >$, we have $< (ab)c >, < (ab)e >, < ace >, < bce >$, and of these, both $< (ab)e >$ and $< ace >$ are not frequent 3-sequences, so we prune $< (ab)ce >$. Thus, C_4 consists of the sequence $< (ab)(cd) >$.

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.

Question 5.1
You would end up with a tree that contains a lot of nodes. This is because when we order by descending frequency, this puts the most common elements at the top so that they can be reused for future patterns, thus reducing the amount of duplicate nodes. However, if you do ascending instead, then there will be lots of node because not as many can be shared along each path of the tree. There will be lots of nodes with low count values (such as 1 or 2). As a result, this isn't an optimal compressed encoding, and thus will result in a slower algorithm.

Question 5.2
The idea of CloSpan is that it uses closed frequent sequential patterns to do the mining, which

allows it to significantly improve its performance. A closed frequent sequential pattern is a pattern that has no supersequence with the same support. CloSpan then finds these closed sequences that have a support above `min_support`. It then uses a lexicographic sequence tree and applies a projection similar to prefix-span, but cuts down on the time spent on this by pruning the search space. It does this by using the largest prefix possible, thus only mining closed frequent sequential patterns. For example, for the set of data `{<(d)(e)(af)>, <(d)(e)(fg)>}`, it bypasses mining on `(d)` or `(e)` and mines on `(d)(e)` directly. This helps to reduce the cost and number of nodes compared to PrefixSpan, thus increasing its speed. Both PrefixTree and CloSpan are DFS approaches to sequential pattern mining, while GSP is a BFS approach. Also, since CloSpan is an improvement on PrefixSpan, it also doesn't need to generate candidate sequences and also constructs projected databases, and compared to PrefixSpan, it minimizes the number of projected databases formed by mining closed frequent sequential patterns.

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

```
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:
            temp_set = frozenset([item])
            if temp_set not in one_cand_set:
                one_cand_set.add(temp_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.
    Update set_count_map.
    """
    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
```

```
68
69      #========================#
70      # STRART YOUR CODE HERE  #
71      #========================#
72      for item, count in local_set.items():
73          if count >= min_support:
74              rtn.add(item)
75      #========================#
76      #   END YOUR CODE HERE   #
77      #========================#
78
79      return rtn, set_count_map
80
81
82  def run_apriori_loops(one_cand_set, set_count_map, all_transactions,
83                        min_support):
84      """
85      Return:
86          freq_map: a dict
87              {<length_of_set_l>: <set_of_frequent_itemsets_of_length_l>}
88          set_count_map: updated set_count_map
89      """
90      freq_map = dict()
91      current_l_set = one_cand_set
92      i = 1
93      #========================#
94      # STRART YOUR CODE HERE  #
95      #========================#
96      while (current_l_set != set([])):
97          freq_map[i] = current_l_set
98          current_l_set = join_set(current_l_set, i)
99          current_c_set, set_count_map = get_items_with_min_support(current_l_set, all_transactions, min_support, set_count_map)
100         current_l_set = current_c_set
101         i += 1
102     #========================#
103     #   END YOUR CODE HERE   #
104     #========================#
105
106     return freq_map, set_count_map
107
108
109 def get_frequent_items(set_count_map, freq_map):
110     """ Return frequent items as a list. """
111     rtn_items = []
112     for key, value in freq_map.items():
113         rtn_items.extend(
114             [(tuple(item), get_support(set_count_map, item))
115              for item in value])
116     return rtn_items
117
118
119 def get_frequent_rules(set_count_map, freq_map, min_conf):
120     """ Return frequent rules as a list. """
121     rtn_rules = []
122     for key, value in islice(freq_map.items(),1,None):
123         for item in value:
124             _subsets = map(frozenset, [x for x in subsets(item)])
125             for element in _subsets:
126                 remain = item.difference(element)
127                 if len(remain) > 0:
128                 #========================#
129                 # STRART YOUR CODE HERE  #
130
131                 #========================#
132                     both = element.union(remain)
133                     confidence = set_count_map[both] / set_count_map[element]
134                 #========================#
135                 #   END YOUR CODE HERE   #
136                 #========================#
```

```
                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 = 0
    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 item1 in s:
        for item2 in s:
            if item1 is not item2:
                # union must be of length l+1 for this to work
                # otherwise, we're generating a set that's too long
                combined = item1.union(item2)
                if len(combined) == l+1:
                    join_set.add(frozenset(combined))

    #=======================#
    #   END YOUR CODE HERE   #
    #=======================#
    return join_set


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()
```