Khantil Choksi (khchoksi), Aman Chauhan (achauha3)

ALDA CSC 522 HW4

Q1

Import libraries

```
import numpy as np
import pandas as pd
from copy import deepcopy
import matplotlib.pyplot as plt
from scipy.spatial import distance
%matplotlib inline
```

Helper Functions

```
# Helper function to draw scatter plot
def scatter_plot(data, centroids, clusters):
    figure = plt.figure(figsize=(8,8))
    plt.scatter(data[:,0], data[:,1], color='b')
    colors_map = ['blue', 'green', 'red','black']
    plt.xlim(0, 10)
    plt.ylim(0, 10)
    for i, text in enumerate(data_labels):
        plt.annotate(text, (data[i,0]*(1.04), data[i,1]*(1.04)), fontsize =
14)
    for i in range(k):
        plt.scatter(centroids[i,0], centroids[i,1], marker='*',
color=colors_map[i], s=250)
    for i in range(n):
        plt.scatter(data[i,0], data[i,1],color=colors_map[clusters[i]])
    plt.grid(which='both', linestyle='dotted')
    plt.show()
```

```
data = np.array([[1,8], [1,1], [2,4], [3,3], [4,9], [4,6], [6,4], [7,7],
[9,9], [9,1]])
```

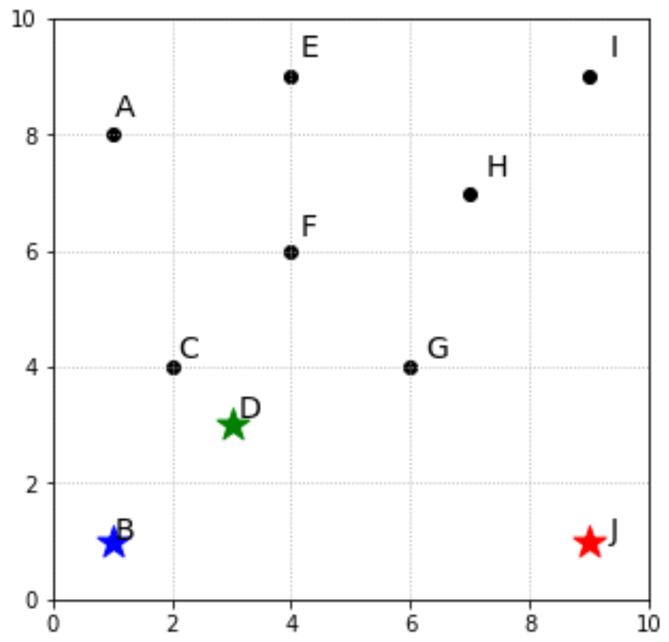
```
data_labels = np.array(['A','B','C','D','E','F','G','H','I','J'])
```

```
# Number of clusters: 3
k = 3
# Number of datapoints: 10
n = 10
```

```
# Initial Seeds / Centroids
centroids = np.array([[1,1], [3,3], [9,1]])
```

```
# Plot points
epoch = 0
print('epoch : '+str(epoch))
figure = plt.figure(figsize=(5,5))
plt.scatter(data[:,0], data[:,1], color='black')
colors_map = ['blue', 'green', 'red']
plt.xlim(0, 10)
plt.ylim(0, 10)
for i, text in enumerate(data_labels):
    plt.annotate(text, (data[i,0]*(1.04), data[i,1]*(1.04)), fontsize = 14)
for i in range(k):
    plt.scatter(centroids[i,0], centroids[i,1], marker='*',
color=colors_map[i], s=250)

plt.grid(which='both', linestyle='dotted')
plt.show()
```

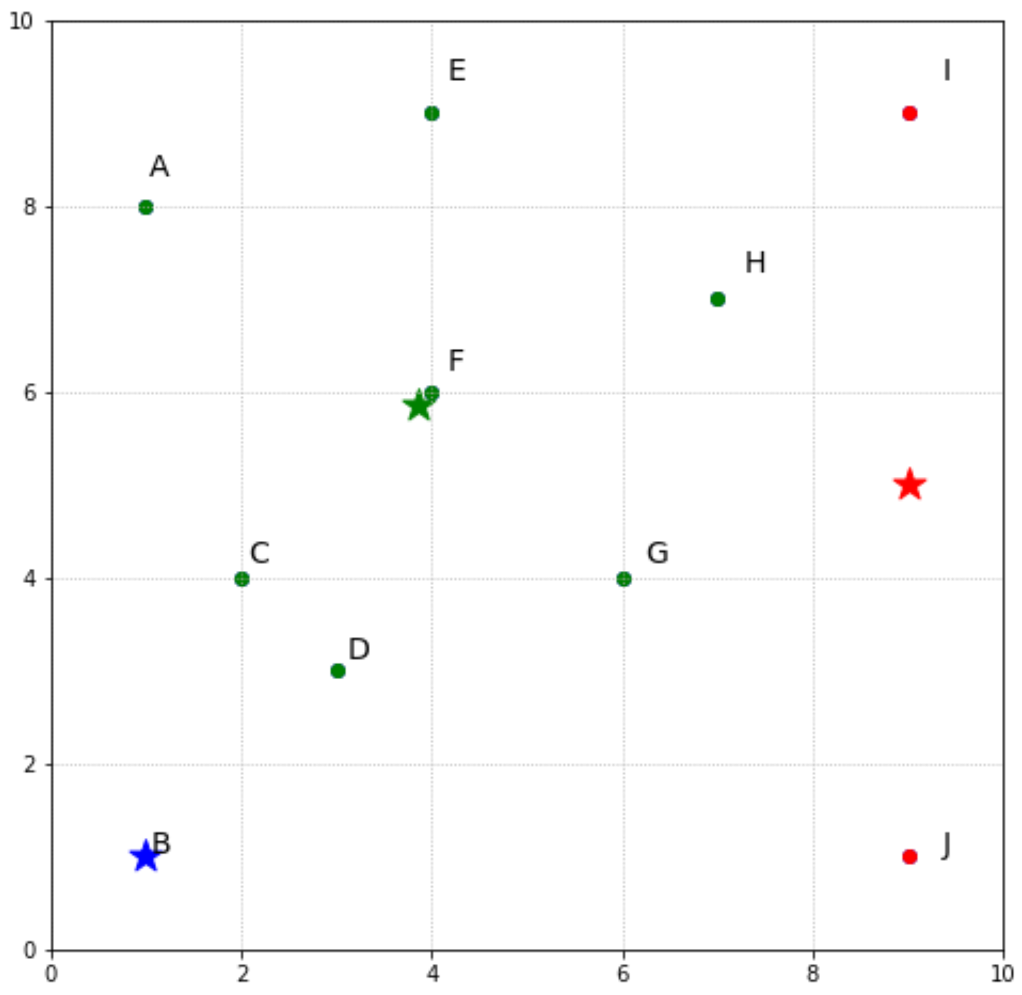
```
epoch : 0
```



(a) Running 1 round of K-means

```
epoch += 1
print('epoch: '+str(epoch))
distances = np.zeros(shape=(n,k))
clusters = np.zeros(n)

new_centroids = np.zeros(centroids.shape)
old_centroids = deepcopy(centroids)


for i in range(k):
    distances[:,i] = np.linalg.norm(data - centroids[i], axis = 1)
clusters = np.argmin(distances, axis = 1)

old_centroids = deepcopy(new_centroids)

# print('\n Clusters: {}'.format(clusters))
# Calculating new centroids
for i in range(k):
    new_centroids[i] = np.mean([list(x) for x,y in zip(data,clusters) if y ==
i], axis=0)


print('Distances from-')
print('\tCluster_0 Centroid \tCluster_1 Centroid \tCluster_2 Centroid')
for i in range(n):
    for j in range(k):
        print ('\t{:.4f}\t\t'.format(distances[i][j]), end='')
    print()

print('----------------------------------------------')
print('Coordinates of the new centroids :-')
for i in range(k):
    print('\tCluster_{} Centroid: '.format(i), end='')
    for j in range(2):
        print ('\t{:.4f}'.format(distances[i][j]), end='')
    print()

print('------------------------------')
print('New Clusters are :-')
for i in range(k):
    print('\tCluster_{} : '.format(i), end='')
    print (' '.join(y for x,y in zip(clusters,data_labels) if x ==i))

scatter_plot(data, new_centroids,clusters)
```

```
epoch: 1
Distances from-
    Cluster_0 Centroid  Cluster_1 Centroid  Cluster_2 Centroid
    7.0000          5.3852          10.6301
    0.0000          2.8284          8.0000
    3.1623          1.4142          7.6158
    2.8284          0.0000          6.3246
    8.5440          6.0828          9.4340
    5.8310          3.1623          7.0711
    5.8310          3.1623          4.2426
    8.4853          5.6569          6.3246
    11.3137         8.4853          8.0000
    8.0000          6.3246          0.0000
----------------------------------------------
Coordinates of the new centroids :-
    Cluster_0 Centroid:     7.0000  5.3852
    Cluster_1 Centroid:     0.0000  2.8284
    Cluster_2 Centroid:     3.1623  1.4142
------------------------------
New Clusters are :-
    Cluster_0 : B
    Cluster_1 : A C D E F G H
    Cluster_2 : I J
```

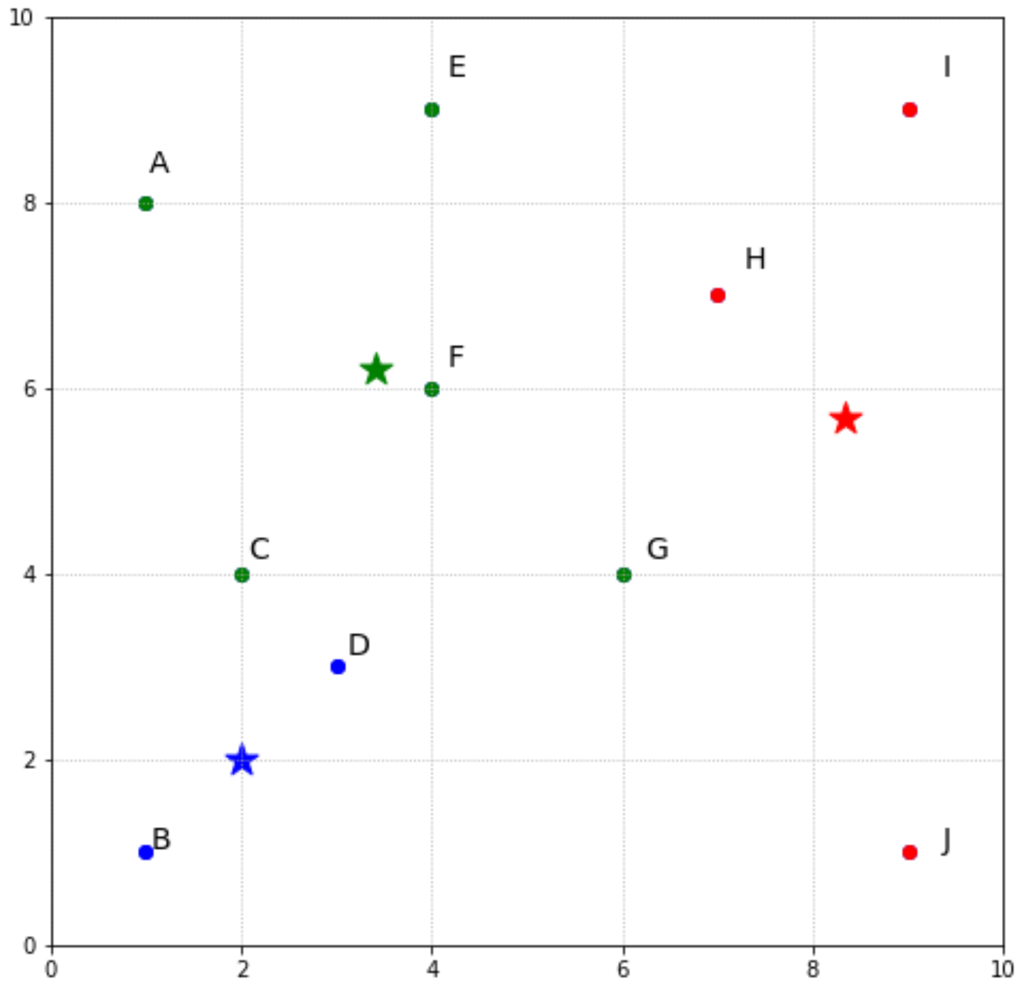


(b) K - means clustering alogrithm rounds

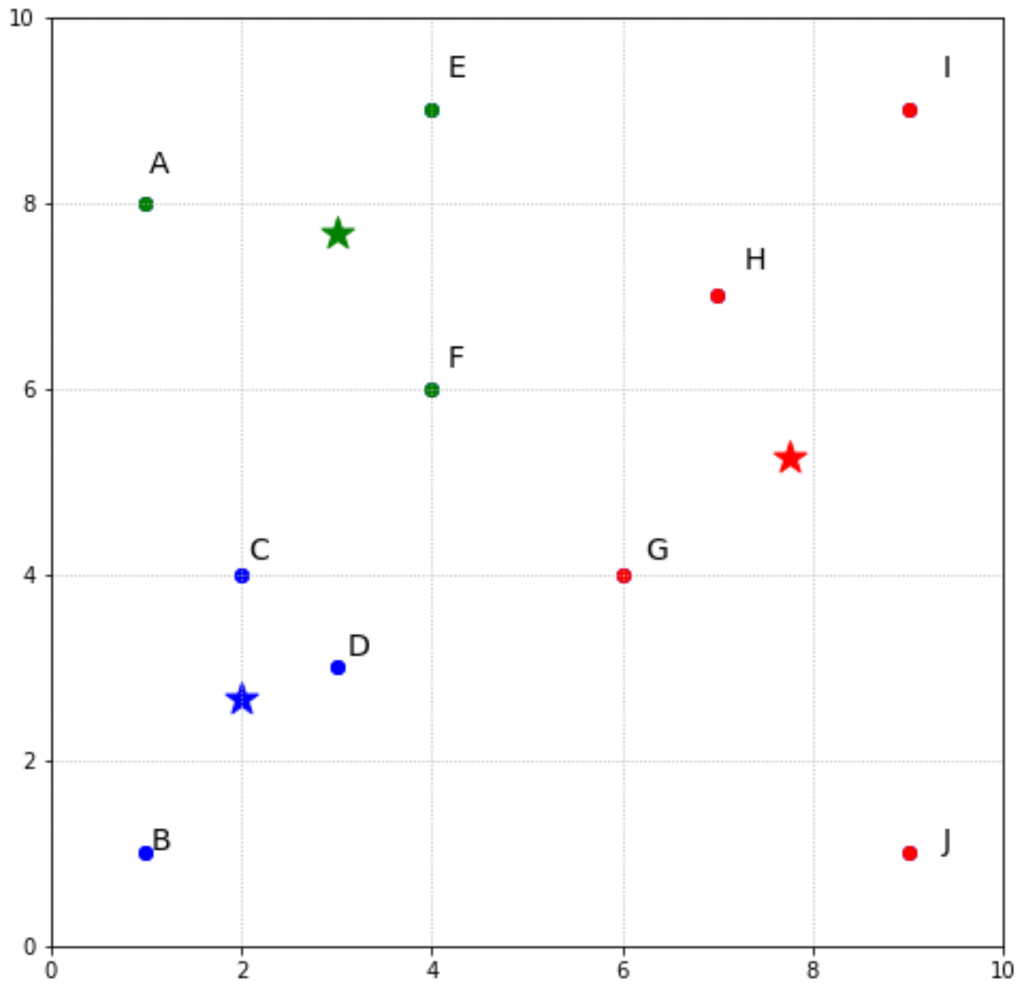
```
epoch = 2
while not np.array_equal(new_centroids, old_centroids) :
    print('--------------------------------------------------------------')
    print('--------------------------------------------------------------')
    print('epoch : '+str(epoch))
#     distances = np.zeros(shape=(n,k))
#     clusters = np.zeros(k)

    for i in range(k):
        distances[:,i] = np.linalg.norm(data - new_centroids[i], axis = 1)
    clusters = np.argmin(distances, axis = 1)

    old_centroids = deepcopy(new_centroids)

    # Calculating new centroids
    # new_centroids[i] = np.mean(data[clusters == i], axis=0)
    for i in range(k):
        # print('Cluster points: {}'.format([list(x) for x,y in zip(data,clusters) if y == i]))
        new_centroids[i] = np.mean([list(x) for x,y in zip(data,clusters) if y == i], axis=0)

    print('Distances from-')
    print('\tCluster_0 Centroid \tCluster_1 Centroid \tCluster_2 Centroid')
    for i in range(n):
        for j in range(k):
            print ('\t{:.4f}\t\t'.format(distances[i][j]), end='')
        print()

    print('------------------------------')
    print('New Clusters are :-')
    for i in range(k):
        print('\tCluster_{} : '.format(i), end='')
        print (' '.join(y for x,y in zip(clusters,data_labels) if x ==i))

    print('----------------------------------------')
    print('Coordinates of the new centroids :-')
    for i in range(k):
        print('\tCluster_{} Centroid: '.format(i), end='')
        for j in range(2):
            print ('\t{:.4f}'.format(new_centroids[i][j]), end='')
        print()

    scatter_plot(data, new_centroids, clusters)
    epoch += 1
```

```
-------------------------------------------------------------
-------------------------------------------------------------
epoch : 2
Distances from-
    Cluster_0 Centroid  Cluster_1 Centroid  Cluster_2 Centroid
    7.0000          3.5714          8.5440
    0.0000          5.6352          8.9443
    3.1623          2.6264          7.0711
    2.8284          2.9829          6.3246
    8.5440          3.1461          6.4031
    5.8310          0.2020          5.0990
    5.8310          2.8356          3.1623
    8.4853          3.3442          2.8284
    11.3137         6.0271          4.0000
    8.0000          7.0740          4.0000
------------------------------
```

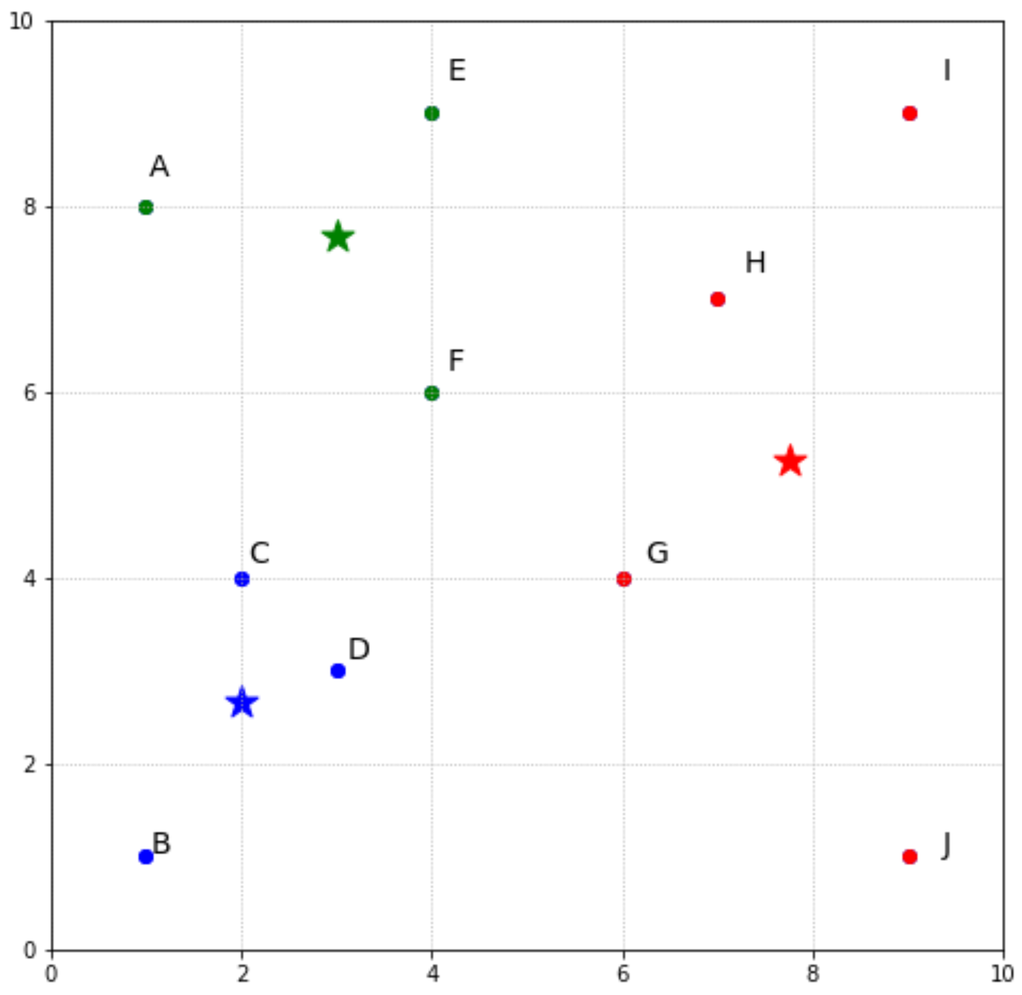
```
New Clusters are :-
    Cluster_0 : B D
    Cluster_1 : A C E F G
    Cluster_2 : H I J
---------------------------------------
Coordinates of the new centroids :-
    Cluster_0 Centroid:     2.0000  2.0000
    Cluster_1 Centroid:     3.4000  6.2000
    Cluster_2 Centroid:     8.3333  5.6667
```



```
-------------------------------------------------------------
-------------------------------------------------------------
epoch : 3
Distances from-
    Cluster_0 Centroid  Cluster_1 Centroid  Cluster_2 Centroid
    6.0828          3.0000          7.6956
    1.4142          5.7271          8.6923
    2.0000          2.6077          6.5490
    1.4142          3.2249          5.9628
    7.2801          2.8636          5.4671
    4.4721          0.6325          4.3461
    4.4721          3.4059          2.8674
    7.0711          3.6878          1.8856
    9.8995          6.2610          3.3993
    7.0711          7.6420          4.7140
------------------------------
New Clusters are :-
    Cluster_0 : B C D
    Cluster_1 : A E F
    Cluster_2 : G H I J
---------------------------------------
Coordinates of the new centroids :-
    Cluster_0 Centroid:     2.0000  2.6667
    Cluster_1 Centroid:     3.0000  7.6667
    Cluster_2 Centroid:     7.7500  5.2500
```



```
-------------------------------------------------------------
-------------------------------------------------------------
epoch : 4
Distances from-
    Cluster_0 Centroid  Cluster_1 Centroid  Cluster_2 Centroid
    5.4263          2.0276          7.2887
    1.9437          6.9602          7.9765
    1.3333          3.8006          5.8843
    1.0541          4.6667          5.2559
    6.6416          1.6667          5.3033
    3.8873          1.9437          3.8243
    4.2164          4.7376          2.1506
    6.6165          4.0552          1.9039
    9.4399          6.1464          3.9528
    7.1957          8.9691          4.4300
------------------------------
New Clusters are :-
    Cluster_0 : B C D
    Cluster_1 : A E F
    Cluster_2 : G H I J
---------------------------------------
Coordinates of the new centroids :-
    Cluster_0 Centroid:     2.0000  2.6667
    Cluster_1 Centroid:     3.0000  7.6667
    Cluster_2 Centroid:     7.7500  5.2500
```



Conclusion:

- Total 3 rounds required for K-means clustering algorithm to converge.
- Here in the figure, cluster's centroid points are indicated with star symbol.
- 3 clusters are denoted with blue, green and red.

For Q2 (c) calculating SSE

```
#print('Distances from-')
#print('\tCluster_0 Centroid \tCluster_1 Centroid \tCluster_2 Centroid')
ss = 0
for i in range(k):
        cluster_points = [list(x) for x,y in zip(data,clusters) if y == i]
        # print('Cluster Points: {}'.format(cluster_points))
        distances = np.linalg.norm(cluster_points - new_centroids[i], axis = 
1)
        ss += sum([x*x for x in distances ])
print('K-Means SSE: {}'.format(ss))
```

```
K-Means SSE: 60.83333333333333
```

Khantil & Aman
HW4

Problem 2 Hierarchical Clustering

Euclidean Distance Matrix:

	A	B	C	D	E	F	G	H	I	J
A	0.00	7.00	4.12	5.39	3.16	3.61	6.40	6.08	8.06	10.63
B		0.00	3.16	2.83	8.54	5.83	5.83	8.49	11.31	8.00
C			0.00	1.41	5.39	2.83	4.00	5.83	8.60	7.62
D				0.00	6.08	3.16	3.16	5.66	8.49	6.32
E					0.00	3.00	5.39	3.61	5.00	9.43
F						0.00	2.83	3.16	5.83	7.07
G							0.00	3.16	5.83	4.24
H								0.00	2.83	6.32
I									0.00	8.00
J										0.00

(a)
Single Link Hierarchical Clustering:
(Similarity of two clusters is based on the two most similar closest points in different clusters)

Step 1: {C} will merge with {D} at 1.41

	A	B	C,D	E	F	G	H	I	J
A	0.00	7.00	4.12	3.16	3.61	6.40	6.08	8.06	10.63
B		0.00	2.83	8.54	5.83	5.83	8.49	11.31	8.00
C,D			**0.00**	**5.39**	**2.83**	**3.16**	**5.66**	**8.49**	**6.32**
E				0.00	3.00	5.39	3.61	5.00	9.43
F					0.00	2.83	3.16	5.83	7.07
G						0.00	3.16	5.83	4.24
H							0.00	2.83	6.32
I								0.00	8.00
J									0.00

Step 2: {C,D} will merge with {B} at 2.83
Here 2.83 distance is same for {C,D} and {B} & {H} and {I}, so I have chosen the first one.

	A	C,D,B	E	F	G	H	I	J
A	0.00	4.12	3.16	3.61	6.40	6.08	8.06	10.63
C,D,B		**0.00**	**5.39**	**2.83**	**3.16**	**5.66**	**8.49**	**6.32**
E			0.00	3.00	5.39	3.61	5.00	9.43
F				0.00	2.83	3.16	5.83	7.07
G					0.00	3.16	5.83	4.24
H						0.00	2.83	6.32
I							0.00	8.00
J								0.00

Step 3: {C,D,B} will merge with {F} at 2.83

	A	C,D,B,F	E	G	H	I	J
A	0.00	3.61	3.16	6.40	6.08	8.06	10.63
C,D,B,F		**0.00**	**3.00**	**2.83**	**3.16**	**5.83**	**6.32**
E			0.00	5.39	3.61	5.00	9.43
G				0.00	3.16	5.83	4.24
H					0.00	2.83	6.32
I						0.00	8.00
J							0.00

Step 4: {C,D,B,F} will merge with {G} 2.83

	A	C,D,B,F,G	E	H	I	J
A	0.00	3.61	3.16	6.08	8.06	10.63
C,D,B,F,G		**0.00**	**3.00**	**3.16**	**5.83**	**4.24**
E			0.00	3.61	5.00	9.43
H				0.00	2.83	6.32
I					0.00	8.00
J						0.00

Step 5: {H} will merge with {I} 2.83

	A	C,D,B,F,G	E	**H,I**	J
A	0.00	3.61	3.16	**6.08**	10.63
C,D,B,F,G		0.00	3.00	**3.16**	4.24
E			0.00	**3.61**	9.43
H,I				**0.00**	**6.32**
J					0.00

Step 5: {C,D,B,F,G} will merge with {E} 3.00

	A	C,D,B,F,G,E	**H,I**	J
A	0.00	3.16	**6.08**	10.63
C,D,B,F,G,E		0.00	**3.16**	4.24
H,I			**0.00**	**6.32**
J				0.00

Step 6: {C,D,B,F,G,E} will merge with {H,I} at height 3.16

	A	C,D,B,F,G,E,H,I	J
A	0.00	3.16	10.63
C,D,B,F,G,E,H,I		**0.00**	**4.24**
J			0.00

Step 7: {C,D,B,F,G,E,H,I} will merge with {A} at height 3.16

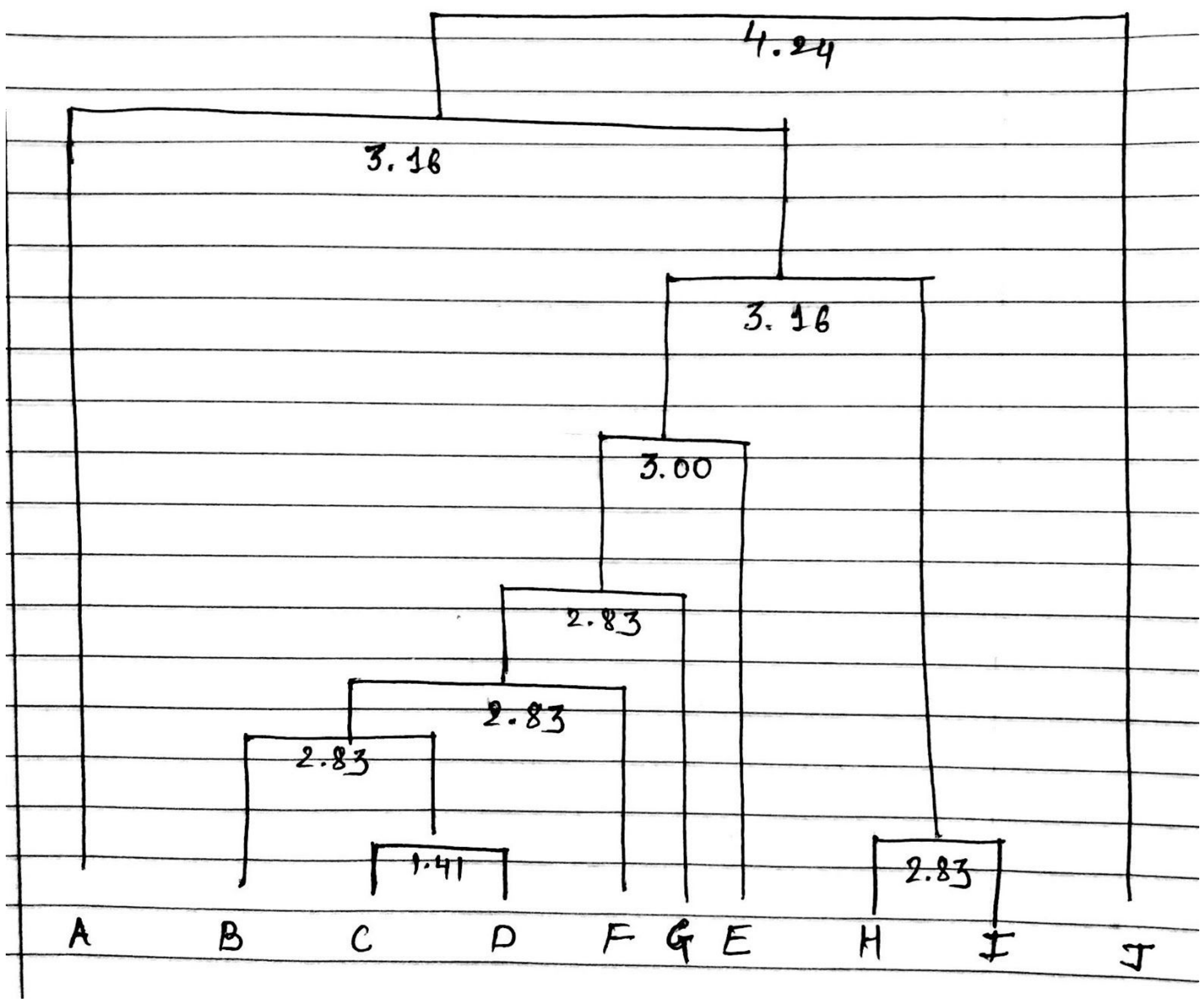
	C,D,B,F,G,E,H,I,A	J
C,D,B,F,G,E,H,I,A	**0.00**	**4.24**
J		0.00

Step 8: {C,D,B,F,G,H,I,A} will merge with {J} at height 4.24

Dendrogram:

Khantil, Aman

Single Link



Complete Link Hierarchical Clustering:
(Similarity of two clusters is based on the two **least** similar (most distant) points in different clusters)

Step 1: {C} will merge with {D} at 1.41

	A	B	**C,D**	E	F	G	H	I	J
A	0.00	7.00	**5.39**	3.16	3.61	6.40	6.08	8.06	10.63
B		0.00	**3.16**	8.54	5.83	5.83	8.49	11.31	8.00
C,D			**0.00**	**6.08**	**3.16**	**4.00**	**5.83**	**8.60**	**7.62**
E				0.00	3.00	5.39	3.61	5.00	9.43
F					0.00	2.83	3.16	5.83	7.07
G						0.00	3.16	5.83	4.24
H							0.00	2.83	6.32
I								0.00	8.00
J									0.00

Step 2: {G} will merge with {F} at 2.83

	A	B	C,D	E	**G,F**	H	I	J
A	0.00	7.00	5.39	3.16	6.40	6.08	8.06	10.63
B		0.00	3.16	8.54	**5.83**	8.49	11.31	8.00
C,D			0.00	6.08	4.00	5.83	8.60	7.62
E				0.00	5.39	3.61	5.00	9.43
G,F					**0.00**	**3.16**	**5.83**	**7.07**
H						0.00	2.83	6.32
I							0.00	8.00
J								0.00

Step 4: {H} will merge with {I} at 2.83

	A	B	C,D	E	G,F	**H,I**	J
A	0.00	7.00	5.39	3.16	6.40	**8.06**	10.63
B		0.00	3.16	8.54	5.83	**11.31**	8.00
C,D			0.00	6.08	4.00	**8.60**	7.62
E				0.00	5.39	**5.00**	9.43
G,F					0.00	**5.83**	7.07
H,I						**0.00**	**8.00**
J							0.00

Step 5: {A} will merge with {E} at 3.16

	A,E	B	C,D	G,F	H,I	J
A,E	**0.00**	**8.54**	**5.39**	**6.40**	**8.06**	**10.63**
B		0.00	3.16	5.83	11.31	8.00
C,D			0.00	4.00	8.60	7.62
G,F				0.00	5.83	7.07
H,I					0.00	8.00
J						0.00

Step 6: {B} will merge with {C,D} at 3.16

	A,E	**B,C,D**	G,F	H,I	J
A,E	0.00	**8.54**	6.40	8.06	10.63
B,C,D		**0.00**	**5.83**	**11.31**	**8.00**
G,F			0.00	5.83	7.07
H,I				0.00	8.00
J					0.00

Step 7: {B,C,D} will merge with {G,F} at 5.83

	A,E	**B,C,D,F,G**	H,I	J
A,E	0.00	**8.54**	8.06	10.63
B,C,D,F,G		**0.00**	**11.31**	**8.00**
H,I			0.00	8.00
J				0.00

Step 7: {B,C,D,F,G} will merge with {J} at 8.00

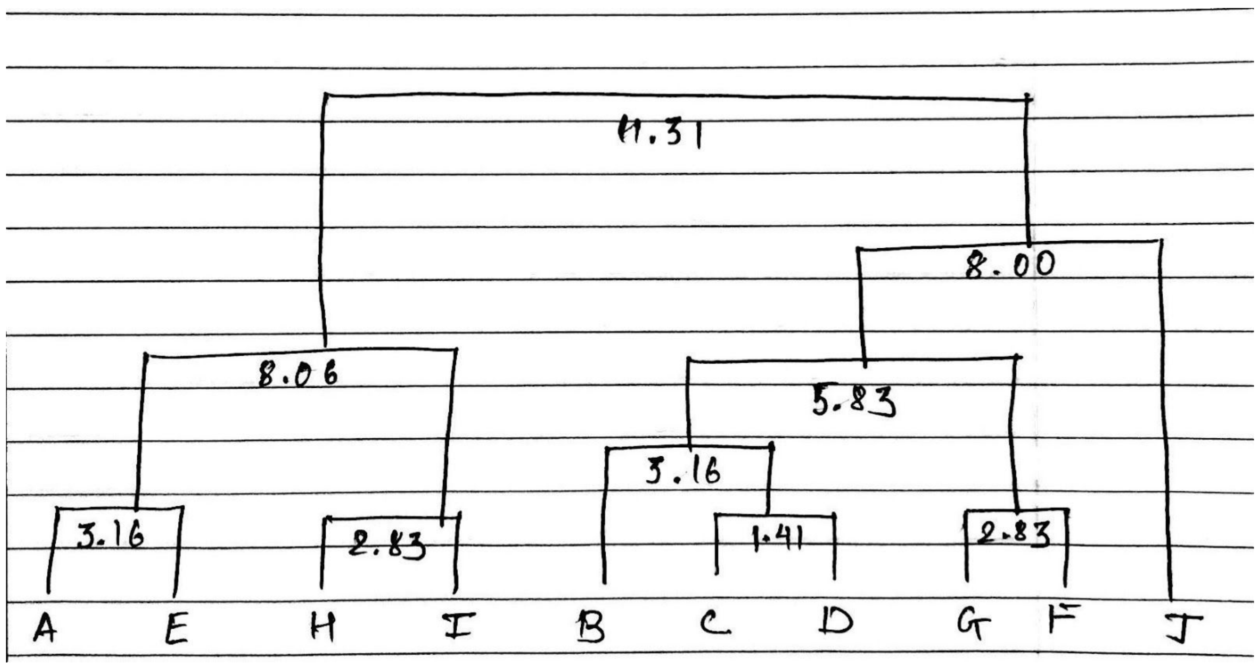
	A,E	**B,C,D,F,G,J**	H,I
A,E	0.00	10.63	8.06
B,C,D,F,G,J		**0.00**	**11.31**
H,I			0.00

Step 8: {A,E} will merge with {H,I} at 8.06

	A,E,H,I	**B,C,D,F,G,J**
A,E,H,I	0.00	**11.31**
B,C,D,F,G,J		**0.00**

Step 9: {A,E,H,I} will merge with {B,C,D,F,G,J} at 11.31

Dendrogram:



(b) If we assume there are three clusters, which of the single and complete link hierarchical clustering will give better resulted clusters?
Single Link Hierarchical:
3 Clusters: {B,C,D,E,G,H,I} , {J} and {A}
Single Link SSE: 107.875 ((Calculated in Q2.pynb)

Complete Link Hierarchical:
3 Clusters: {B,C,D,G,F} , {J} and {A,E,H,I}
SSE: 72.0 (Calculated in Q2.pynb)

As we can visualize from dendrogram and cluster formation, Single link tends to provide elongated clusters whereas complete link short and circular clusters.
Conclusion: If there are three clusters, based on Sum of Squared Error calculated and dendrogram, Complete link gives better result.

(c) Compare your resulted clusters from 2(b) with the resulted clusters using K-means in Question 1 by calculating their corresponding Sum of Squared Error (SSE). Based on their SSE results, which resulted clusters, 1(b) or 2(b), are better?

From Q1, SSE for K-Means Clustering is : 60.83333333333333 (Calculated in Q1.pynb)

Comparing results with Q1(b) and Q2(b),
K-Means provide better results, as it provides least Sum of Squared Errors.

Q2

Import Libraries

```
import math
import numpy as np
import pandas as pd
from copy import deepcopy
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial import distance
%matplotlib inline
```

```
# number of datapoints
n = 10
data = np.array([[1,8], [1,1], [2,4], [3,3], [4,9], [4,6], [6,4], [7,7],
[9,9], [9,1]])
data_labels = np.array(['A','B','C','D','E','F','G','H','I','J'])
```

```
k = n
distances = np.zeros(shape=(n,n))
for i in range(n):
    distances[:,i] = np.linalg.norm(data - data[i], axis = 1)
print('Distances from-')
clusters = np.zeros(k)
clusters = deepcopy(data)
clusters_label = deepcopy(data_labels)
for cluster_label in clusters_label:
    print('\t{}'.format(cluster_label), end='')
print()
for i in range(k):
    print('{}'.format(clusters_label[i]), end='')
    for j in range(k):
        print ('\t{:.4f}'.format(distances[i][j]), end='')
    print()
```

```
Distances from-
    A    B    C    D    E    F    G    H    I    J
A   0.0000  7.0000  4.1231  5.3852  3.1623  3.6056  6.4031  6.0828  8.0623
10.6301
B   7.0000  0.0000  3.1623  2.8284  8.5440  5.8310  5.8310  8.4853  11.3137
8.0000
C   4.1231  3.1623  0.0000  1.4142  5.3852  2.8284  4.0000  5.8310  8.6023
7.6158
D   5.3852  2.8284  1.4142  0.0000  6.0828  3.1623  3.1623  5.6569  8.4853
6.3246
E   3.1623  8.5440  5.3852  6.0828  0.0000  3.0000  5.3852  3.6056  5.0000
9.4340
F   3.6056  5.8310  2.8284  3.1623  3.0000  0.0000  2.8284  3.1623  5.8310
7.0711
G   6.4031  5.8310  4.0000  3.1623  5.3852  2.8284  0.0000  3.1623  5.8310
4.2426
H   6.0828  8.4853  5.8310  5.6569  3.6056  3.1623  3.1623  0.0000  2.8284
6.3246
I   8.0623  11.3137 8.6023  8.4853  5.0000  5.8310  5.8310  2.8284  0.0000
8.0000
J   10.6301 8.0000  7.6158  6.3246  9.4340  7.0711  4.2426  6.3246  8.0000
0.0000
```

```
# Single Link
# Cluster 0 {B,C,D,E,F,G,H,I}
# Cluster 1 {J}
# Cluster 2 {A}
k = 3
single_link_centroids = np.zeros(shape=(k,2))
clusters = np.zeros(10)
clusters = [2,0,0,0,0,0,0,0,0,1]

# Calculating new centroids
# new_centroids[i] = np.mean(data[clusters == i], axis=0)
for i in range(k):
    # print('Cluster points: {}'.format([list(x) for x,y in zip(data,clusters)
if y == i]))
    single_link_centroids[i] = np.mean([list(x) for x,y in zip(data,clusters)
if y == i], axis=0)

ss = 0
for i in range(k):
        cluster_points = [list(x) for x,y in zip(data,clusters) if y == i]
        # print('Cluster Points: {}'.format(cluster_points))
        distances = np.linalg.norm(cluster_points - single_link_centroids[i], 
axis = 1)
        ss += sum([x*x for x in distances ])
print('Single Link SSE: {}'.format(ss))
```

```
Single Link SSE: 107.875
```

```
# Complete Link
# Cluster 0 {A,B,C,D,F,G,E}
# Cluster 1 {J}
# Cluster 2 {H,I}
k = 3
single_link_centroids = np.zeros(shape=(k,2))
clusters = np.zeros(10)
clusters = [0,0,0,0,0,0,0,2,2,1]

# Calculating new centroids
# new_centroids[i] = np.mean(data[clusters == i], axis=0)
for i in range(k):
    # print('Cluster points: {}'.format([list(x) for x,y in zip(data,clusters)
if y == i]))
    single_link_centroids[i] = np.mean([list(x) for x,y in zip(data,clusters)
if y == i], axis=0)

ss = 0
for i in range(k):
        cluster_points = [list(x) for x,y in zip(data,clusters) if y == i]
        # print('Cluster Points: {}'.format(cluster_points))
        distances = np.linalg.norm(cluster_points - single_link_centroids[i], 
axis = 1)
        ss += sum([x*x for x in distances ])
print('Complete Link SSE: {}'.format(ss))
```

```
Complete Link SSE: 72.0
```

Q3

Import Libraries

```
from pprint import pprint
import itertools
import math
```

(a) Explain what is frequent itemset and give an example of 2-itemset that is frequent itemset with support count = 8.

Frequent Itemset: Itemset is collection of one or more itemsets. Frequent itemset is whose support is greater than or equal to a minimum support threshold defined.

```
# Transactions
transactions = [(1, frozenset({'B','D','F','H'})),
                (2, frozenset({'C','D','F','G'})),
                (3, frozenset({'A','D','F','G'})),
                (4, frozenset({'A','B','C','D','H'})),
                (5, frozenset({'A','C','F','G'})),
                (6, frozenset({'D','H'})),
                (7, frozenset({'A','B','E','F'})),
                (8, frozenset({'A','D','F','G','H'})),
                (9, frozenset({'A','C','D','F','G'})),
                (10, frozenset({'D','F','G','H'})),
                (11, frozenset({'A','C','D','E'})),
                (12, frozenset({'B','E','F','H'})),
                (13, frozenset({'D','F','G'})),
                (14, frozenset({'C','F','G','H'})),
                (15, frozenset({'A','C','D','F','H'})),
               ]
```

```
# unique items
items = set()
for i, x in transactions:
    items.update(x)
print('Total items in the data set - {}'.format(len(items)))
print('The items are - {}'.format(items))
```

```
Total items in the data set - 8
The items are - {'B', 'C', 'A', 'G', 'H', 'D', 'F', 'E'}
```

```
# Itemsets with support >= 1
itemsets = {}

for x in itertools.combinations(items, 2):
    for _, y in transactions:
        if set(x).issubset(y):
            if frozenset(x) not in itemsets:
                itemsets[frozenset(x)]=0
            itemsets[frozenset(x)]+=1

itemsets_support8 = dict((itemset,count)for itemset, count in itemsets.items()
if count >= 8)
s = set(itemsets_support8.keys())
print('Itemsets with support = 8 are as following: ')
for x in s:
   print('[{}]'.format(', '.join(x)))
print('Total itemsets with support = 8 are ->
{}'.format(len(itemsets_support8)))
```

```
Itemsets with support = 8 are as following:
[F, G]
[D, F]
Total itemsets with support = 8 are -> 2
```

(b) Explain what is closed frequent itemset and list all of them with supportcount = 8.

Closed Frequent Itemset: It is a frequent itemset that is both closed and its support is greater than or equal to minsup. An itemset is closed in a data set if there exists no superset that has the same support count as this original itemset. Ref: http://www.hypertextbookshop.com/dataminingbook/public_version/contents/chapters/chapter002/section004/blue/page002.html

```
itemsets = {}
for i in range(1, len(items)+1):
    for x in itertools.combinations(items, i):
        for _, y in transactions:
            if set(x).issubset(y):
                if frozenset(x) not in itemsets:
                        itemsets[frozenset(x)]=0
                    itemsets[frozenset(x)]+=1
itemsets

itemsets_support8_dict = dict((itemset,count)for itemset, count in
itemsets.items() if count >= 8)
itemsets_support8 = list(itemsets_support8_dict.keys())

# Find closed itemset
closed_frequent_itemsets = []
for i in range(len(itemsets_support8)):
    subsetFound = False
    for j in range(i+1,len(itemsets_support8)):

        if itemsets_support8[i].issubset(itemsets_support8[j]) and
itemsets_support8_dict[itemsets_support8[i]] <=
itemsets_support8_dict[itemsets_support8[j]]:
            subsetFound = True
            break

    if not subsetFound:
        closed_frequent_itemsets.append(itemsets_support8[i])

# print('itemsets_support8_dict: {}'.format(itemsets_support8_dict))
# s = set(itemsets_support8_dict.keys())
# for x in s:
#    print('[{}]'.format(', '.join(x)))

print('Closed Itemsets with support >= 8 are as following: ')
for x in closed_frequent_itemsets:
   print('[{}]'.format(', '.join(x)))
# print('Closed itemsets with support >= 8 are ->
{}'.format(len(itemsets_support8)))
```

```
Closed Itemsets with support >= 8 are as following:
[A]
[H]
[D]
[F]
[F, G]
[D, F]
```

(c) Explain what is maximal frequent itemset and list all of maximal itemset with support count = 8.

Maximal Frequent Itemset: It is a frequent itemset for which none of its immediate supersets are frequent. Ref: http://www.hypertextbookshop.com/dataminingbook/public_version/contents/chapters/chapter002/section004/blue/page001.html

```
itemsets = {}
for i in range(1, len(items)+1):
    for x in itertools.combinations(items, i):
        for _, y in transactions:
            if set(x).issubset(y):
                if frozenset(x) not in itemsets:
                    itemsets[frozenset(x)]=0
                itemsets[frozenset(x)]+=1
itemsets

itemsets_support8_dict = dict((itemset,count)for itemset, count in
itemsets.items() if count >= 8)
itemsets_support8 = list(itemsets_support8_dict.keys())

# Find closed itemset
maximal_frequent_itemsets = []
for i in range(len(itemsets_support8)):
    subsetFound = False
    for j in range(i+1,len(itemsets_support8)):
        if itemsets_support8[i].issubset(itemsets_support8[j]) and
len(itemsets_support8[i])+1 == len(itemsets_support8[j]):
                subsetFound = True
                break

    if not subsetFound:
        maximal_frequent_itemsets.append(itemsets_support8[i])

# s = set(itemsets_support8_dict.keys())
# for x in s:
#    print('[{}]'.format(', '.join(x)))

print('Maximal Frequent Itemsets with support >= 8 are as following: ')
for x in maximal_frequent_itemsets:
   print('[{}]'.format(', '.join(x)))
# print('Closed itemsets with support >= 8 are ->
{}'.format(len(itemsets_support8)))
```

```
Maximal Frequent Itemsets with support >= 8 are as following:
[A]
[H]
[F, G]
[D, F]
```

(d) Compute the support and confidence for association rule {D, F} -> {G}

```python
# Support: Fraction of transactions that contains {D,F,G}
association_rule_set = frozenset({'D','F','G'})
support = 0
# for i in range(len(association_rule_set), len(items)+1):
#     for x in itertools.combinations(items, i):
for _,t in transactions:
    # print('Transaction: {}'.format(t))
    if association_rule_set.issubset(t):
        support += 1
print('Support for association rule {{D, F}} -> {{G}} is
{}'.format(support/len(transactions)))

# Confidence
x_set = frozenset({'D','F'})
support_x = 0
for _,t in transactions:
    # print('Transaction: {}'.format(t))
    if x_set.issubset(t):
        support_x += 1
print('Confidence for association rule {{D, F}} -> {{G}} is
{}'.format(support/support_x))
```

```
Support for association rule {D, F} -> {G} is 0.4
Confidence for association rule {D, F} -> {G} is 0.75
```

Q4 - Association Analysis

Consider the following market basket transactions shown in the Table below.

Transaction ID	Items Ordered
1	{Flour, Eggs, Bread}
2	{Soda, Coffee}
3	{Flour, Butter, Milk, Eggs}
4	{Bread, Eggs, Juice, Detergent}
5	{Bread, Milk, Eggs}
6	{Eggs, Bread}
7	{Detergent, Milk}
8	{Coffee, Soda, Juice}
9	{Butter, Juice, Bread}
10	{Milk, Bread, Detergent}

Import Libraries

```python
from pprint import pprint
import itertools
import math
```

(a) How many items are in this data set? What is the maximum size of itemsets that can be extracted from this data set?

Transaction History

```
transactions = [(1, frozenset({'Flour', 'Eggs', 'Bread'})),
                (2, frozenset({'Soda', 'Coffee'})),
                (3, frozenset({'Flour', 'Butter', 'Milk', 'Eggs'})),
                (4, frozenset({'Bread', 'Eggs', 'Juice', 'Detergent'})),
                (5, frozenset({'Bread', 'Milk', 'Eggs'})),
                (6, frozenset({'Eggs', 'Bread'})),
                (7, frozenset({'Detergent', 'Milk'})),
                (8, frozenset({'Coffee', 'Soda', 'Juice'})),
                (9, frozenset({'Butter', 'Juice', 'Bread'})),
                (10, frozenset({'Milk', 'Bread', 'Detergent'})),
               ]
```

Count of Items

```
items = set()
for i, x in transactions:
    items.update(x)
print('Total items in the data set - {}'.format(len(items)))
print('The items are - {}'.format(items))
```

```
Total items in the data set - 9
The items are - {'Juice', 'Flour', 'Detergent', 'Eggs', 'Bread', 'Milk',
'Soda', 'Butter', 'Coffee'}
```

Itemsets with Support ≥ 1

```
itemsets = {}
for i in range(1, len(items)+1):
    for x in itertools.combinations(items, i):
        for _, y in transactions:
            if set(x).issubset(y):
                if frozenset(x) not in itemsets:
                    itemsets[frozenset(x)]=0
                itemsets[frozenset(x)]+=1
print('Total itemsets with support ≥ 1 are - {}'.format(len(itemsets)))
```

```
Total itemsets with support ≥ 1 are - 44
```

```
print('Maximum Size of Itemset - {}'.format(max([len(x) for x in
itemsets.keys()])))
```

```
Maximum Size of Itemset - 4
```

There are 9 unique items, and 44 itemsets with with support ≥ 1, with maximum number of items in an itemset being 4.

(b) What is the maximum number of association rules that can be extracted from this data (including rules that have zero support)?

```
cnt=math.pow(3,len(items))-math.pow(2,len(items)+1)+1
print('The maximum number of association rules that can be extracted are {:.0f}'.format(cnt))
```

```
The maximum number of association rules that can be extracted are 18660
```

The maximum number of association rules that can be generated using d distinct items is given by the following formula - $R = \sum_{k=1}^{d-1}[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}] = 3^d - 2^{d+1} + 1$**. If we plugin** $d = 9, R = 18660.$

(c) What is the maximum number of 2-itemsets that can be derived from this data set (including those have zero support)?

```
print('The maximum number of 2-itemsets that can be derived from this data set are {:.0f}'.format(
    math.factorial(len(items))/(math.factorial(len(items)-2)*2)))
```

```
The maximum number of 2-itemsets that can be derived from this data set are
36
```

The maximum number of 2-itemsets that can be derived using n distinct items is given by the following combination - $count = \binom{n}{2} = \frac{n!}{n!2!} = \frac{n(n-1)}{2}$**. For 9 items, we get 36 2-itemsets.**

(d) Find an itemset (of size 2 or larger) that has the largest support.

```python
max_cnt=0
max_set=None
for x,cnt in itemsets.items():
    if len(x)>=2 and cnt>max_cnt:
        max_cnt=cnt
        max_set=x
print('The itemset of size 2 or larger with maximum support is -> {' + ', '.join(max_set) + '}')
```

```
The itemset of size 2 or larger with maximum support is -> {Bread, Eggs}
```

We can calculate this by iterating over the all the itemsets, and checking itemsets which have support more than or equal to 2. We can then maintain a MAX variable to keep track of the set which has the maximum support. Using this approach we get the following itemset -> {Bread, Eggs} and its support is 4 or $\frac{4}{10} = 0.4$ **.**

(e) Given minconf = 0.5, find two pairs of items, a and b, such that the rules {a} -> {b} and {b} -> {a} have the same confidence, and their confidence is greater than or equal to the minconf threshold.

```python
s = set(itemsets.keys())
for x, y in itertools.combinations(s, 2):
    union = x.union(y)
    if union in itemsets and itemsets[x]==itemsets[y] and itemsets[union]/itemsets[x]>=0.5:
        print('Confidence -> {:.2f}\t'.format(itemsets[union]/itemsets[x]),
              '[{}]'.format(', '.join(x)),
              '[{}]'.format(', '.join(y)))
```

```
Confidence -> 1.00   [Butter, Eggs] [Milk, Butter, Eggs]
Confidence -> 1.00   [Butter, Eggs] [Flour, Milk, Eggs]
Confidence -> 1.00   [Butter, Eggs] [Flour, Milk, Butter, Eggs]
Confidence -> 1.00   [Butter, Eggs] [Flour, Milk]
Confidence -> 1.00   [Butter, Eggs] [Flour, Butter, Eggs]
Confidence -> 1.00   [Butter, Eggs] [Flour, Butter]
Confidence -> 1.00   [Butter, Eggs] [Flour, Milk, Butter]
Confidence -> 1.00   [Butter, Eggs] [Milk, Butter]
Confidence -> 1.00   [Milk, Butter, Eggs] [Flour, Milk, Eggs]
Confidence -> 1.00   [Milk, Butter, Eggs] [Flour, Milk, Butter, Eggs]
Confidence -> 1.00   [Milk, Butter, Eggs] [Flour, Milk]
Confidence -> 1.00   [Milk, Butter, Eggs] [Flour, Butter, Eggs]
Confidence -> 1.00   [Milk, Butter, Eggs] [Flour, Butter]
Confidence -> 1.00   [Milk, Butter, Eggs] [Flour, Milk, Butter]
Confidence -> 1.00   [Milk, Butter, Eggs] [Milk, Butter]
Confidence -> 1.00   [Eggs, Detergent] [Bread, Juice, Eggs]
Confidence -> 1.00   [Eggs, Detergent] [Bread, Juice, Detergent]
Confidence -> 1.00   [Eggs, Detergent] [Juice, Eggs]
Confidence -> 1.00   [Eggs, Detergent] [Eggs, Juice, Detergent]
Confidence -> 1.00   [Eggs, Detergent] [Eggs, Bread, Detergent]
Confidence -> 1.00   [Eggs, Detergent] [Juice, Detergent]
Confidence -> 1.00   [Eggs, Detergent] [Eggs, Juice, Bread, Detergent]
Confidence -> 0.50   [Milk, Eggs] [Flour]
Confidence -> 0.50   [Milk, Eggs] [Bread, Milk]
Confidence -> 0.50   [Milk, Eggs] [Flour, Eggs]
Confidence -> 0.50   [Milk, Eggs] [Butter]
Confidence -> 1.00   [Flour] [Flour, Eggs]
Confidence -> 0.50   [Flour] [Butter]
Confidence -> 1.00   [Bread, Juice, Eggs] [Bread, Juice, Detergent]
Confidence -> 1.00   [Bread, Juice, Eggs] [Juice, Eggs]
Confidence -> 1.00   [Bread, Juice, Eggs] [Eggs, Juice, Detergent]
Confidence -> 1.00   [Bread, Juice, Eggs] [Eggs, Bread, Detergent]
Confidence -> 1.00   [Bread, Juice, Eggs] [Juice, Detergent]
Confidence -> 1.00   [Bread, Juice, Eggs] [Eggs, Juice, Bread, Detergent]
Confidence -> 1.00   [Juice, Coffee] [Juice, Soda, Coffee]
Confidence -> 1.00   [Juice, Coffee] [Juice, Soda]
Confidence -> 1.00   [Bread, Juice, Detergent] [Juice, Eggs]
Confidence -> 1.00   [Bread, Juice, Detergent] [Eggs, Juice, Detergent]
Confidence -> 1.00   [Bread, Juice, Detergent] [Eggs, Bread, Detergent]
Confidence -> 1.00   [Bread, Juice, Detergent] [Juice, Detergent]
Confidence -> 1.00   [Bread, Juice, Detergent] [Eggs, Juice, Bread,
Detergent]
Confidence -> 1.00   [Juice, Eggs] [Eggs, Juice, Detergent]
Confidence -> 1.00   [Juice, Eggs] [Eggs, Bread, Detergent]
Confidence -> 1.00   [Juice, Eggs] [Juice, Detergent]
Confidence -> 1.00   [Juice, Eggs] [Eggs, Juice, Bread, Detergent]
Confidence -> 1.00   [Eggs, Juice, Detergent] [Eggs, Bread, Detergent]
Confidence -> 1.00   [Eggs, Juice, Detergent] [Juice, Detergent]
Confidence -> 1.00   [Eggs, Juice, Detergent] [Eggs, Juice, Bread,
Detergent]
Confidence -> 1.00   [Juice, Soda, Coffee] [Juice, Soda]
Confidence -> 1.00   [Flour, Bread, Eggs] [Flour, Bread]
Confidence -> 1.00   [Eggs, Bread, Detergent] [Juice, Detergent]
Confidence -> 1.00   [Eggs, Bread, Detergent] [Eggs, Juice, Bread, Detergent]
Confidence -> 1.00   [Flour, Milk, Eggs] [Flour, Milk, Butter, Eggs]
Confidence -> 1.00   [Flour, Milk, Eggs] [Flour, Milk]
Confidence -> 1.00   [Flour, Milk, Eggs] [Flour, Butter, Eggs]
Confidence -> 1.00   [Flour, Milk, Eggs] [Flour, Butter]
Confidence -> 1.00   [Flour, Milk, Eggs] [Flour, Milk, Butter]
Confidence -> 1.00   [Flour, Milk, Eggs] [Milk, Butter]
Confidence -> 1.00   [Soda, Coffee] [Soda]
Confidence -> 1.00   [Soda, Coffee] [Coffee]
Confidence -> 1.00   [Flour, Milk, Butter, Eggs] [Flour, Milk]
Confidence -> 1.00   [Flour, Milk, Butter, Eggs] [Flour, Butter, Eggs]
Confidence -> 1.00   [Flour, Milk, Butter, Eggs] [Flour, Butter]
Confidence -> 1.00   [Flour, Milk, Butter, Eggs] [Flour, Milk, Butter]
Confidence -> 1.00   [Flour, Milk, Butter, Eggs] [Milk, Butter]
Confidence -> 1.00   [Soda] [Coffee]
Confidence -> 1.00   [Bread, Juice, Butter] [Juice, Butter]
Confidence -> 1.00   [Bread, Juice, Butter] [Bread, Butter]
Confidence -> 0.50   [Bread, Juice] [Bread, Detergent]
Confidence -> 0.50   [Bread, Juice] [Butter]
Confidence -> 0.50   [Milk, Detergent] [Bread, Detergent]
Confidence -> 0.50   [Milk, Detergent] [Bread, Milk]
Confidence -> 0.50   [Bread, Detergent] [Bread, Milk]
Confidence -> 1.00   [Juice, Detergent] [Eggs, Juice, Bread, Detergent]
Confidence -> 0.50   [Flour, Eggs] [Butter]
Confidence -> 1.00   [Flour, Milk] [Flour, Butter, Eggs]
Confidence -> 1.00   [Flour, Milk] [Flour, Butter]
Confidence -> 1.00   [Flour, Milk] [Flour, Milk, Butter]
Confidence -> 1.00   [Flour, Milk] [Milk, Butter]
Confidence -> 1.00   [Juice, Butter] [Bread, Butter]
Confidence -> 1.00   [Flour, Butter, Eggs] [Flour, Butter]
Confidence -> 1.00   [Flour, Butter, Eggs] [Flour, Milk, Butter]
Confidence -> 1.00   [Flour, Butter, Eggs] [Milk, Butter]
Confidence -> 1.00   [Flour, Butter] [Flour, Milk, Butter]
Confidence -> 1.00   [Flour, Butter] [Milk, Butter]
Confidence -> 1.00   [Flour, Milk, Butter] [Milk, Butter]
```

The formula for confidence is given by $confidence(X->Y) = \frac{support(X \cup Y)}{support(X)}$. Given 2 itemsets, x and y, the confidence(x->y) and confidence(y->x) is equal if and only if the their individual supports are equal.

We listed out above all the possible pair of itemsets have a confidence ≥ 0.5 and have the same confidence.

If we assume a and b are items, then the question states to find 2 such pairs of 1-itemsets. From the above list these are [{Flour} {Butter}] and [{Soda}{Coffee}].

If we assume a and b are 2-itemsets, then the questions asks us to find a pair of 2-itemsets that satisfy the above condition. From the above list, this is [{Butter, Eggs}, {Flour, Milk}]

Q5

Import Libraries

```
from pprint import pprint
import itertools
import math
```

```
# Transactions
transactions = [(1, frozenset({'A','C','D','E'})),
                (2, frozenset({'A','B','D','E'})),
                (3, frozenset({'C','E'})),
                (4, frozenset({'C','D'})),
                (5, frozenset({'A','B','D'})),
                (6, frozenset({'B','D','E'})),
                (7, frozenset({'A','C','D'})),
                (8, frozenset({'B','C','D','E'})),
               ]
```

```
# unique items
items = set()
for i, x in transactions:
    items.update(x)
# print('Total items in the data set - {}'.format(len(items)))
print('The unique items are - {}'.format(items))
```

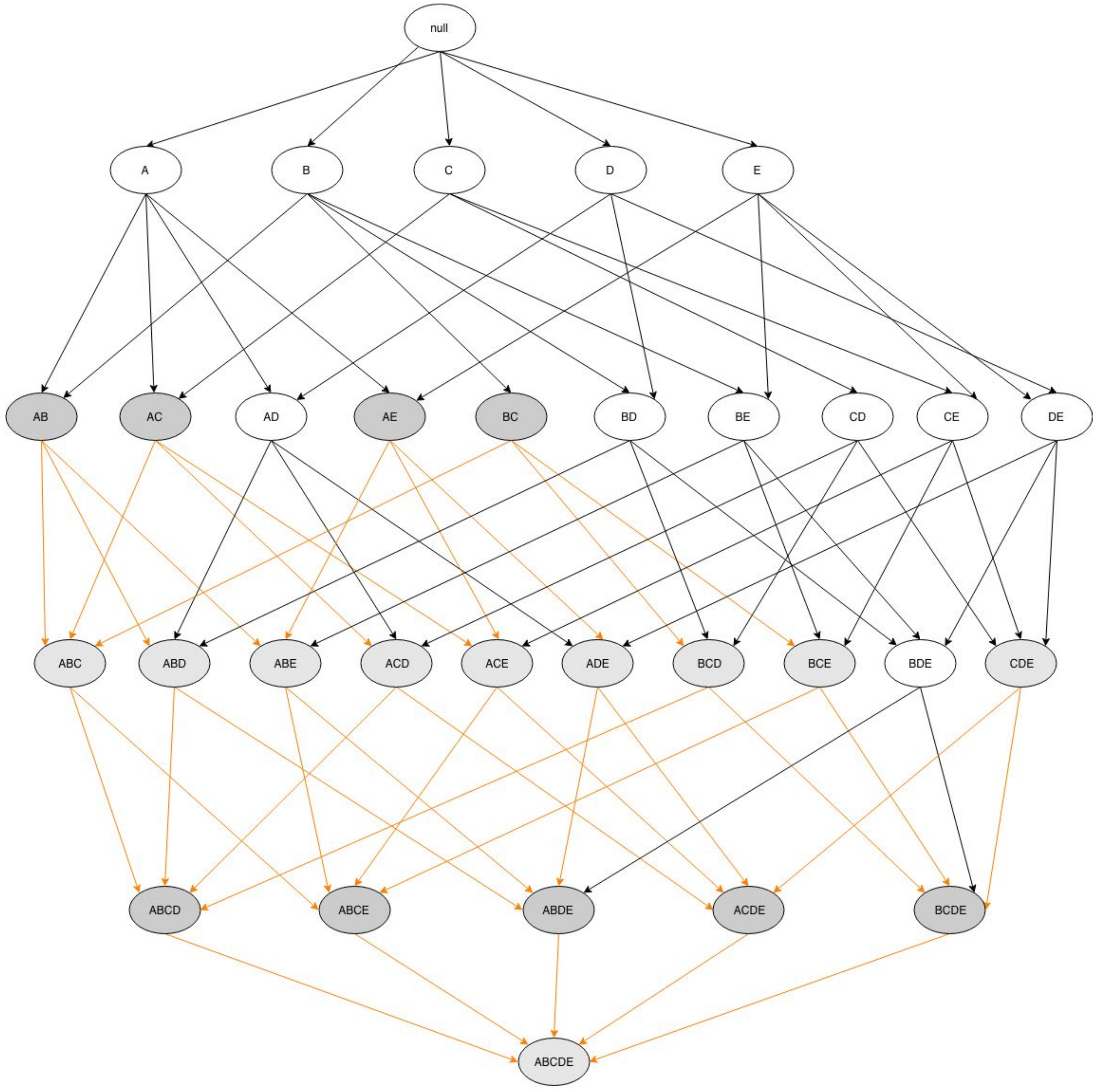
```
The unique items are - {'E', 'A', 'D', 'C', 'B'}
```

```
# Apriori Algorithm
# L1
itemsets = {}
for x in itertools.combinations(items, 1):
    for _, y in transactions:
        if set(x).issubset(y):
            if frozenset(x) not in itemsets:
                itemsets[frozenset(x)]=0
            itemsets[frozenset(x)]+=1
```

```
# Candidate Set
print('Candidate Set C{0}'.format(1))
print('\t Itemset \t Support Count')
for x in itemsets.keys():
    print('\t[{}]'.format(', '.join(x)), '\t {}'.format(itemsets[x]))

itemsets_support3 = dict((itemset,count)for itemset, count in itemsets.items()
if count >= 3)
if len(itemsets_support3) < len(itemsets):
    print('Pruning')
else:
    print('Pruning Not Required.')
itemsets_support3 = dict((itemset,count)for itemset, count in itemsets.items()
if count >= 3)
s = set(itemsets_support3.keys())
print('Step 1 L1')
print('1 - Itemsets with support = 3 are as following: ')
for x in s:
   print('[{}]'.format(', '.join(x)), ' Support Count:
{}'.format(itemsets_support3[x]))

##################################

for i in range(2, len(items)+1):
    if len(itemsets_support3) == 0:
        print('No new frequent itemsets identified')
        break
    print('---------------------------------------------------------------')
    print('---------------------------------------------------------------')
    print('Step: {0} L({0})'.format(i))
    itemsets = {}
    # Take only frequent itemsets for next step, prune step of Apriori
    items = set(itemsets_support3.keys())
    for x1,x2 in itertools.combinations(items, 2):
        # Union Step
        next_frequent_itemset = set(x1.union(x2))
        if frozenset(next_frequent_itemset) not in itemsets:
            for _, y in transactions:
                # print('next_frequent_itemset:
{}'.format(next_frequent_itemset))
                if len(next_frequent_itemset) == i and
next_frequent_itemset.issubset(y):
                    # print('next_frequent_itemset:
{}'.format(next_frequent_itemset))
                    if frozenset(next_frequent_itemset) not in itemsets:
                        itemsets[frozenset(next_frequent_itemset)]=0
                    itemsets[frozenset(next_frequent_itemset)]+=1

    # Candidate Set
    print('Candidate Set C{0}'.format(i))
    print('\t Itemset \t Support Count')
    for x in itemsets.keys():
        print('\t[{}]'.format(', '.join(x)), '\t {}'.format(itemsets[x]))

    itemsets_support3 = dict((itemset,count)for itemset, count in
itemsets.items() if count >= 3)
    if len(itemsets_support3) < len(itemsets):
        print('Pruning')
    else:
        print('Pruning Not Required.')

    print('L{0} Itemsets with support = 3 are as following: '.format(i))
    print('\tItemset \tSupport Count')
    for x in itemsets_support3.keys():
        print('\t[{}]'.format(', '.join(x)), '\t\t
{}'.format(itemsets_support3[x]))
```

```
Candidate Set C1
     Itemset     Support Count
    [E]      5
    [A]      4
    [D]      7
    [C]      5
    [B]      4
Pruning Not Required.
Step 1 L1
1 - Itemsets with support = 3 are as following:
[B]  Support Count: 4
[A]  Support Count: 4
[D]  Support Count: 7
[C]  Support Count: 5
[E]  Support Count: 5
--------------------------------------------------------------
--------------------------------------------------------------
Step: 2 L(2)
Candidate Set C2
     Itemset     Support Count
    [B, A]   2
    [D, B]   4
    [C, B]   1
    [E, B]   3
    [D, A]   4
    [C, A]   2
    [E, A]   2
    [C, D]   4
    [E, D]   4
    [C, E]   3
Pruning
L2 Itemsets with support = 3 are as following:
    Itemset     Support Count
    [D, B]       4
    [E, B]       3
    [D, A]       4
    [C, D]       4
    [E, D]       4
    [C, E]       3
-------------------------------------------------------------
-------------------------------------------------------------
Step: 3 L(3)
Candidate Set C3
     Itemset     Support Count
    [E, D, A]    2
    [D, B, A]    2
    [C, D, A]    2
    [C, E, B]    1
    [E, D, B]    3
    [C, E, D]    2
    [C, D, B]    1
Pruning
L3 Itemsets with support = 3 are as following:
    Itemset     Support Count
    [E, D, B]        3
-------------------------------------------------------------
-------------------------------------------------------------
Step: 4 L(4)
Candidate Set C4
     Itemset     Support Count
Pruning Not Required.
L4 Itemsets with support = 3 are as following:
    Itemset     Support Count
No new frequent itemsets identified
```

Q5(b)
Lattice Structure with pruning



Note:
Nodes shown in grey are pruned.
Branched show in orange will be pruned.

Q6 - FP-Tree

Consider the following data set shown in Table 4 and answer the following questions using FP-Tree.

TID	Items Bought
T1	{B,D,F,H}
T2	{C,D,F,G}
T3	{A,D,F,G}
T4	{A,B,C,D,H}
T5	{A,C,F,G}
T6	{D,H}
T7	{A,B,E,F}
T8	{A,D,F,G,H}
T9	{A,C,D,F,G}
T10	{D,F,G,H}
T11	{A,C,D,E}
T12	{B,E,F,H}
T13	{D,F,G}
T14	{C,F,G,H}
T15	{A,C,D,F,H}

(a) Construct an FP-tree for the set of transactions in the table below as the first step towards identifying the itemsets with minimum support count of 2 (at least 2 occurrences). Do not forget to include the header table that locates the starts of the corresponding linked item lists through the FP-tree. For consistency, please form your header table in the order of {F, D, G, H, A, C, B, E}

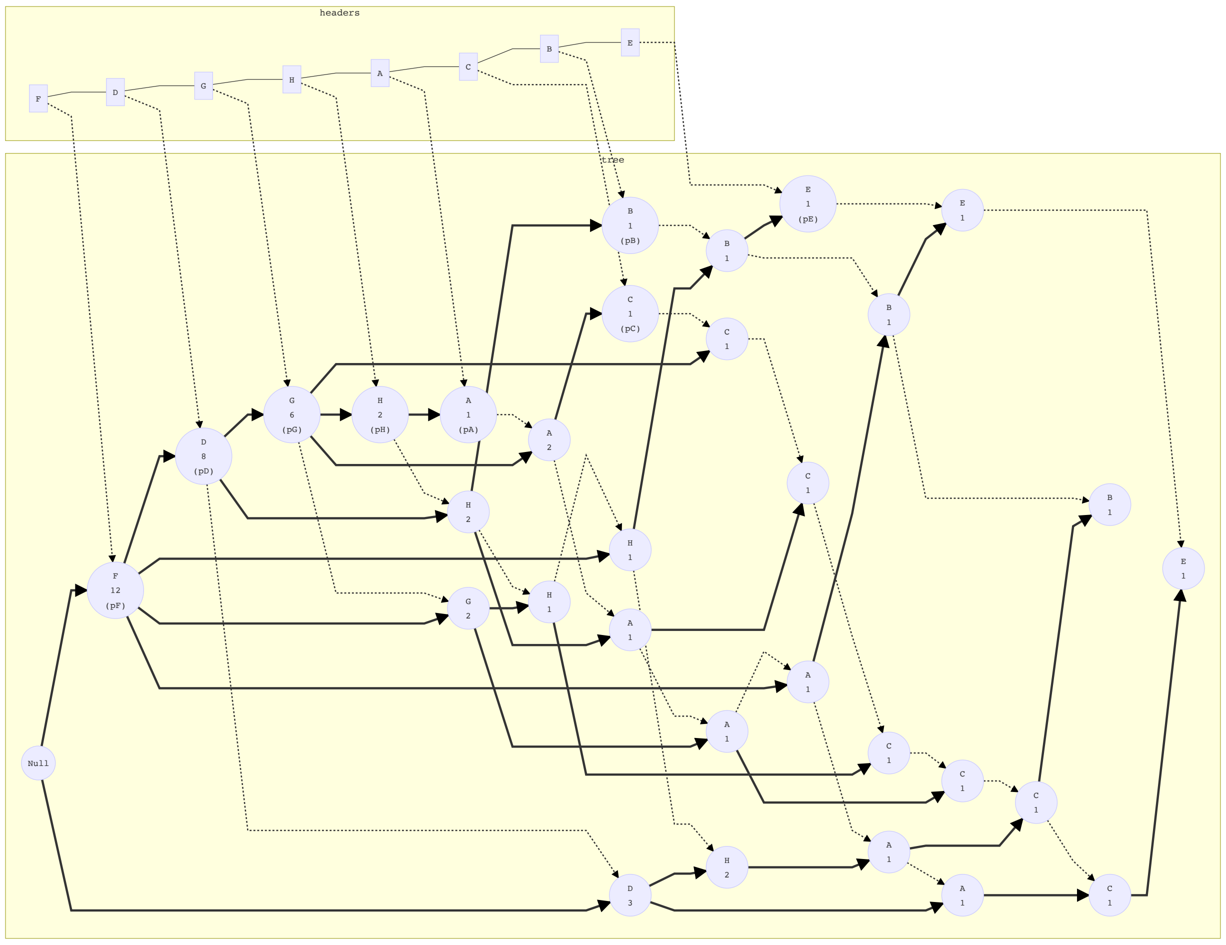
Support Counts / Header Table

Item	Support
F	12
D	11
G	8
H	8
A	8
C	7
B	4
E	3

Ordered Itemsets

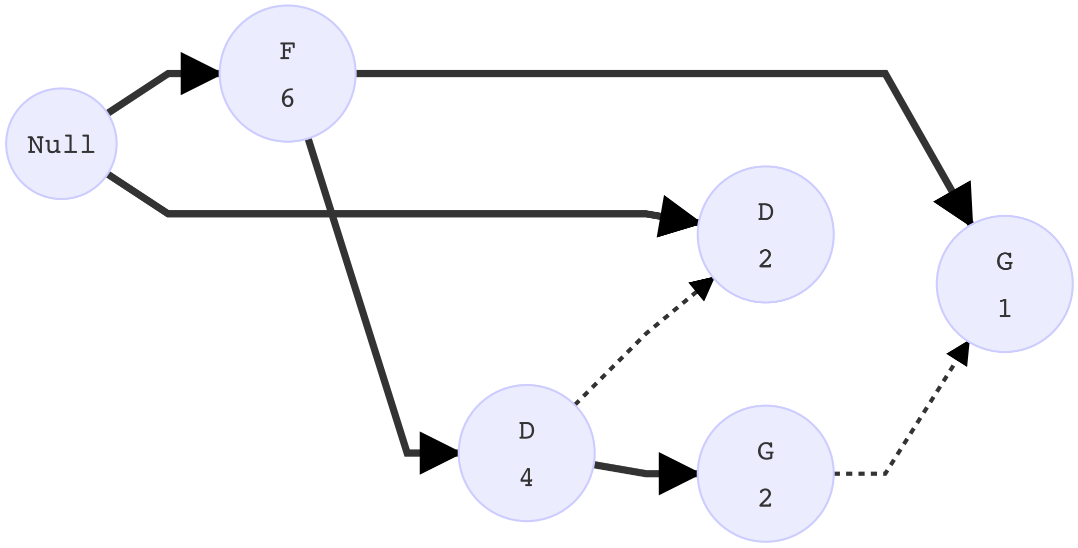
TID	Items Bought	Ordered ItemSets
T1	{B,D,F,H}	{F,D,H,B}
T2	{C,D,F,G}	{F,D,G,C}
T3	{A,D,F,G}	{F,D,G,A}
T4	{A,B,C,D,H}	{D,H,A,C,B}
T5	{A,C,F,G}	{F,G,A,C}
T6	{D,H}	{D,H}
T7	{A,B,E,F}	{F,A,B,E}
T8	{A,D,F,G,H}	{F,D,G,H,A}
T9	{A,C,D,F,G}	{F,D,G,A,C}
T10	{D,F,G,H}	{F,D,G,H}
T11	{A,C,D,E}	{D,A,C,E}
T12	{B,E,F,H}	{F,H,B,E}
T13	{D,F,G}	{F,D,G}
T14	{C,F,G,H}	{F,G,H,C}
T15	{A,C,D,F,H}	{F,D,H,A,C}

Graph


headers
F
D
G
H
A
C
B
E
tree
Null
F 12 (pF)
D 8 (pD)
G 6 (pG)
H 2 (pH)
A 1 (pA)
B 1 (pB)
C 1 (pC)
E 1 (pE)
A 2
H 2
G 2
H 1
H 1
A 1
B 1
C 1
E 1
B 1
C 1
A 1
A 1
C 1
C 1
C 1
B 1
E 1
D 3
H 2
A 1
A 1
C 1

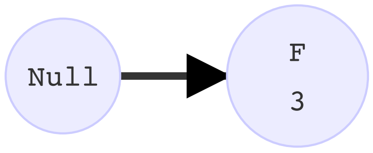
(b) Using the FP-Tree constructed and support=3, generate all the frequent patterns with the base of item H step by step.

(i) Conditional FP-Tree with Base H



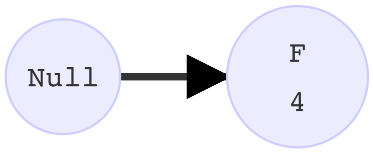
F, D and G are frequent, so we generate itemsets {F,H}, {D,H} and {G,H}.

(ii) Conditional FP-Tree with Base GH



Only F is frequent, so we generate itemset {F,G,H}

(iii) Conditional FP-Tree with Base DH



F is frequent, so we generate itemset {F,D,H}

(iv) Conditional FP-Tree with base FH

Only Null node remains, so no new itemsets.

As H was frequent too, we generate itemset {H} also. Hence, we have the following frequent patterns - **{H}, {F,H}, {D,H}, {G,H}, {F,D,H}, {F,G,H}**.