Q1

(a)
Yes. It can be adapted.

First of all, we obtain the 2-sequence by scanning the database.
(NOTE: The 2-sequence contains two kinds of sequences – (1) the sequence contains only one timestamp entry (e.g. <{D, E}>) and (2) the sequence contains two or more timestamp entries (e.g. <{D}, {E}> ).

The Apriori-like algorithm is described as follows.
1. \( k=2 \)
2. Find all frequent 2-sequences and store them in \( L_k \)
3. repeat
4. \( k=k+1 \)
5. Generate candidate \( k \)-sequences from \( L_{k-1} \) (which will be described later) and store them in \( C_k \)
6. Scan the database and count the support of each candidate in \( C_k \)
7. Find the \( k \)-sequence in \( C_k \) with support \( \geq \) minsupport and store them in \( L_k \)
8. until \( L_k = \text{empty set} \)
9. return \( L_i \) for \( i=2,\ldots,k \)

e.g.
We obtain the following 2-sequences.
\{<\{A\}, \{D\}>, <\{A\}, \{E\}>, <\{A\}, \{G\}>, <\{D\}, \{E\}> \}

Next, we generate the candidate 3-sequences by the join-and-prune process.

The join step of the generation process is described as follows.
A sequence \( s^{(1)} \) is joined with another sequence \( s^{(2)} \) only if the subsequence obtained by dropping the first item in \( s^{(1)} \) is identical to the subsequence obtained by dropping the last item in \( s^{(2)} \). The resulting candidate is the sequence \( s^{(1)} \), concatenated with the last item from \( s^{(2)} \). The last item from \( s^{(2)} \) can either be joined into the same timestamp element as the last item in \( s^{(1)} \) or different timestamp elements depending on the following conditions.

1. If the last two items in \( s^{(2)} \) belong to the same timestamp element, then the last item in \( s^{(2)} \) is part of the last timestamp element in \( s^{(1)} \) in the joined sequence. (e.g. Suppose we have frequent sequences <\{A\}, \{B\}, \{C\}> and <\{B\}, \{C\}, \{D\}> in \( L_{k-1} \). Candidate <\{A\}, \{B\}, \{C, D\}> is obtained by joining <\{A\}, \{B\}, \{C\}> and <\{B\}, \{C\}, \{D\}>).
2. If the last two items in \( s^{(2)} \) belong to different timestamp elements, then the last item in \( s^{(2)} \) becomes a separate timestamp element appended to the end of \( s^{(1)} \) in the joined sequence. (e.g. Suppose we have frequent sequences <\{A\}, \{B\}, \{C\}> and <\{B\}, \{C\}, \{D\}> in \( L_{k-1} \). Candidate <\{A\}, \{B\}, \{C\}, \{D\}> is obtained by joining <\{A\}, \{B\}, \{C\}> and <\{B\}, \{C\}, \{D\}>).

e.g. In the running example, we obtain one candidate 3-sequence <\{A\}, \{D\}, \{E\}> (by joining <\{A\}, \{D\}> and <\{D\}, \{E\}>) after the join step.

The prune step of the generation process is described as follows.
A candidate \( k \)-sequence is pruned if at least one of its \((k-1)\)-sequence is infrequent.
For example, <\{A\}, \{D\}, \{E\}> is a candidate 3-sequence. We need to check whether <\{A\}, \{E\}> is a frequent 2-sequence (NOTE: We do not need to check whether <\{A\}, \{D\}> and <\{D\}, \{E\}> are frequent 2-sequence
because \(<\{A\}, \{D,E\}\>\) was generated from these two frequent sequences). Since \(<\{A\}, \{E\}\>\) is frequent, \(<\{A\}, \{D,E\}\>\) is also considered as a candidate 3-sequence after the prune step.

Then, we do the counting step to count the support of each candidate in the set. As the support of \(<\{A\}, \{D,E\}\>\) is 2, then it is one of the final results.

We repeat the process until \(L_k\) is an empty set.

In our running example, all sequences with support at least 2 are \(<\{A\}, \{D,E\}\>\)

(b) No. It cannot be adapted.
This is because the Apriori property cannot be satisfied.

Consider the following example containing three sequences for three customers.

\(<\{A\}, \{B\}, \{B\}, \{C\}\>\)
\(<\{A\}, \{B\}\>\)
\(<\{A\}, \{B\}\>\)

The support of a 2-sequence \(<\{A\}, \{B\}\>\) is equal to \(4/3 = 1.33\).
The support of a 3-sequence \(<\{A\}, \{B\}, \{C\}\>\) is equal to \(2/1 = 2\).
Since the 3-sequence \(<\{A\}, \{B\}, \{C\}\>\) can be derived from 2-sequence \(<\{A\}, \{B\}\>\) by appending one element at the end (with a new timestamp), and the support of this 3-sequence is larger than the support of this 2-sequence, the Apriori property cannot be satisfied.

Q2.

(a) i. \(F_c=\{<\{A\},3>,<\{C\},3>,<\{D\},4>,<\{C,D\},3>\}\)
   ii. \(C_c=\{<\{A\},3>,<\{D\},4>,<\{C,D\},3>\}\)
   iii. \(M_c=\{<\{A\},3>,<\{C,D\},3>\}\)

(b) i. Advantages:
   1. The number of possible frequent closed itemsets is much smaller than that of frequent itemsets. Therefore, the storage is smaller.
   2. We do not need to generate all possible frequent itemsets. Therefore, execution time is shorter.

Disadvantages:
1. We need to expand the closed frequent itemset in order to check whether an itemset is frequent or not.
2. It is not easy for users to understand the closed frequent itemsets directly.

ii. Advantages:
1. According to closed frequent itemssets with \(C_c\), we can generate all traditional frequent itemsets associated with frequencies (i.e. \(F_c\)). However, according to maximal frequent itemsets with \(M_c\), we cannot generate all traditional frequent itemsets associated with frequencies (i.e. \(F_c\)).
2. Closed frequent itemsets are useful to remove some of the redundant association rules. According to the concept of closed frequent itemsets, we can define redundant association rules as follows. \(X \rightarrow Y\) is redundant if there exists another rule \(X' \rightarrow Y'\) where \(X' \subset X\) and
Y' ⊆ Y, and their support and confidence are the same. Such redundant rules are not generated if closed frequent itemsets are used for rule generation.

Disadvantages:
1. The number of possible closed frequent itemsets is much larger than that of frequent itemsets. So the storage is larger.
2. It is not easy for users to understand the closed frequent itemsets directly.

(c) We can use the same FP-growth approach to mine closed frequent itemset. We just need to make the following adaptions.
1. When we generate conditional FP-trees, we process the items in the header table from the bottom to the top.
2. Whenever we generate an itemset, we have to check whether this itemset is in the set of itemsets generated before.

Algorithm FP-growth (Tree, α)
1. A ← ∅; B ← ∅
2. If tree contains a single path P
   • Partition the path into different segments such that all nodes in each segment have the same node support
   • For each segment S
     o Choose the node N in S at the deepest height
     o β ← a set of nodes N to the root
     o Generate the pattern β ∪ α with support equal to minimum support of nodes in β
     o If there does NOT exist an itemset I in A s.t. I.count = (β ∪ α).count,
       Insert β ∪ α into A
     else
     • For each a_i in the header of Tree,
       o Generate pattern β = a_i ∪ α with support equal to a_i .support
       o Insert this pattern into B
       o Construct β’s conditional pattern base and then β’s conditional FP-tree Tree_β
       o If Tree_β ≠ ∅
         Call FP-growth(Tree_β, β)
3. Last Step
   (a) Process itemsets in B in descending order of the size of the itemsets
   (b) For each itemset I_B in B,
   (c) Check whether there exists an itemset I_A in A s.t. I_A is a super-itemset of I_B and I_A .count = I_B .count. If no, insert I_B into A
   (d) Output A.
E.g.:

<table>
<thead>
<tr>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 3</td>
</tr>
<tr>
<td>B 1</td>
</tr>
<tr>
<td>C 3</td>
</tr>
<tr>
<td>D 4</td>
</tr>
</tbody>
</table>

Sorted:

<table>
<thead>
<tr>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>D 4</td>
</tr>
<tr>
<td>A 3</td>
</tr>
<tr>
<td>C 3</td>
</tr>
</tbody>
</table>

Trans: D,C

A
D, C
D, A, C
A
D

Then, conditional tree on C is:

Trans:
D:1, A:1, C:1
D:2, C:2

B={<{C},3>}
A={<{C,D},3>}

Conditional tree on A is:

Trans:
D:1, A:1
A:2

B={<{C},3>,<{A},3>}
A={<{C,D},3>}
Conditional tree on $D$ is:

Trans:
$D$: 4

\[
\begin{array}{c|c}
\text{D} & \text{freq} \\
\hline
D & 4 \\
\end{array}
\]

$B = \{\{C\}, 3\}, \{\{A\}, 3\}, \{\{D\}, 4\}\}$

$A = \{\{C,D\}, 3\}\}$

Last Step:
Finally, $A = \{\{C,D\}, 3\}, \{\{D\}, 4\}, \{\{A\}, 3\}\}$

Q3

(a)

(i) Cluster1: mean=(59.2857, 59.5714) points: x1, x2, x3, x4, x5, x6, x7
Cluster2: mean=(60.0000, 80.0000) points: x8

(ii) Cluster1: mean=(54.2500, 58.0000) points: x2, x4, x6, x7
Cluster2: mean=(64.5000, 66.2500) points: x1, x3, x5, x8

(iii) Cluster1: mean=(64.5, 59) points: x1, x3, x5, x7
Cluster2: mean=(60, 80) points: x8
Cluster3: mean=(52.33, 60.33) points: x2, x4, x6

(iv) Cluster1: mean=(60, 51) points: x7
Cluster2: mean=(60, 80) points: x8
Cluster3: mean=(52.3333, 60.3333) points: x2, x4, x6
Cluster4: mean=(66, 61.6667) points: x1, x3, x5

(b) No.

The following shows an example with 20 points. The number next to each point denotes the time ID of this point where a small number means an earlier point. Let $k = 2$.

Assume that the initial means are shown as follows.
Consider "original k-means clustering". We have the following output $O_1$.

Consider "sequential k-means clustering". We have the following output $O_2$.

As shown above, $O_1$ is not equal to $O_2$. 
Q4

(a) Distance

Dendrogram

(135) (246) 7 E

(246) 1

7

8

1357) (246) 8

(1357) (246) 8

(135) (246) 7 E

(1234567) 8

(1234567) 8
Consider three data points, 1, 3 and 5. The distance between data point 1 and data point 3, $d_{13}$, is 5. The distance between data point 1 and data point 5, $d_{15}$, is 7. The distance between data point 3 and data point 5, $d_{35}$, is 1. Note that $d_{13} + d_{35}=5+1=6$ and $d_{15} = 7$. Thus, $d_{13}+d_{35} < d_{15}$ which violates the triangle inequality. Thus, there is no metric space satisfying this matrix. Since the $d$-dimensional space using the Euclidean distance is a metric space, it is not possible to have such a $d$-dimensional space.

Note that some students consider the complex system for solving this problem, in which case, the triangle inequality property could be violated.
(a) We could define a distance function \( \text{dist}(., .) \) between any 2 possible values for each categorical attribute \( i \) where the distance value is smaller whenever these 2 values are similar. For example, the distance between “Hong Kong” and “Macau” is smaller than the distance between “Hong Kong” and “New York”.

The distance between a point \((a_1, a_2, ..., a_d)\) and another point \((b_1, b_2, ..., b_d)\) is defined to be
\[
\sum_{i=1}^{d} \text{dist}(a_i, b_i)^2.
\]
Then, we could adopt the major principle of k-medoids.

(b) No. This is because this algorithm terminates when \( k \) is equal to the total number of data points. In this case, \( e_k \) is equal to 0. Even if \( k \) is larger, \( e_k \) is also equal to 0 and thus \( e_k \) converges. The value \( k \) is not what we desire because we want to group “similar” points.

Alg:

a. First, multiply each attribute of each data point by a positive real number \( \Delta \) such that the “closest” pair between two points is at least 1.0.

b. Second, define \( d_k \) to be the product of the distances between any two clusters according to distance “single linkage”.

c. We change Step 4 from \( e_k \) to \( d_k \).

d. We change Step 5 to the following:

We repeat Step 3 to Step 4 for different possible values of \( k \) and obtain the corresponding values of \( d_k \).

e. We find the \( k \) s.t. \( d_k \) is maximized.

The reasons are:

1. If \( k \) is larger than the number of clusters, in k-means, one single cluster will be represented by more than one means. In other words, the real cluster is split into a number of groups in k-means. It is expected that these groups are very close and thus \( d_k \) is small.

2. If \( k \) is smaller than the number of clusters, in k-means, two or more clusters will be represented by one mean/group.

There are two cases:

- Case 1:

![Diagram of Case 1](image)

One cluster is split into two or more groups.

In the example, when \( k=2 \), cluster 2 is split into group 1 and group 2. It is easy to see that \( d_k \) is small.

- Case 2:
In this example, when $k=2$, cluster 1 and cluster 2 completely belong to group 1 while cluster 3 and cluster 4 completely belong to group 2. In this case, we know that the separation between group 1 and group 2 is large and thus $d_k$ is large. However, when $k=4$, we have

![Diagram of clusters and groups]

Obviously, $d_k$ in this case is larger.
In conclusion, if we find $k$ s.t. $d_k$ is maximized, we can find a good value for the number of clusters.