Q1

(a) Yes.

Apriori algorithm for itemsets

Apriori algorithm for items. the table is like this:

(S_1, S_2, S_3, S_4) means itemsets

TID	A	B	C	D	E
t_1	1	0	0	1	0
t_2	1	0	0	1	1
t_3	0	0	1	0	0
t_4	1	0	1	1	1
t_5	1	0	1	0	1

transform item into itemsets →

SequenceID	S_1	S_2	S_3	S_4	S_5
S_A	1	0	0	1	0
S_B	1	0	0	1	1
S_C	0	0	1	0	0
S_D	1	0	1	1	1
S_E	1	0	1	0	1

start from here! ↓

threshold support = 2

L_1:

itemsets	Support
$\{S_1\}$	4
$\{S_2\}$	0
$\{S_3\}$	3
$\{S_4\}$	3
$\{S_5\}$	3

→

C_2:

$\{\{S_1, S_3\}, \{S_1, S_4\}, \{S_1, S_5\}, \{S_3, S_4\}, \{S_3, S_5\}, \{S_4, S_5\}\}$

L_2:

Sequence	Support
$\{S_1, S_3\}$	2
$\{S_1, S_4\}$	3
$\{S_1, S_5\}$	3
$\{S_3, S_4\}$	1
$\{S_3, S_5\}$	2
$\{S_4, S_5\}$	2

→

C_3:

$\{\{S_1, S_3, S_4\}, \{S_1, S_3, S_5\}, \{S_1, S_4, S_5\}\}$

↓

the result is $\{\{S_1, S_3, S_5\}, \{S_1, S_4, S_5\}\}$

S_1, S_2, S_3 ... are the smallest units in this kind of Apriori algorithm.

The Apriori Algorithm is like this:

① $k=2$
② look for 1 2-sequences (which support≥2) and store them in L_2;
③ Loop
④ $k=k++$;
⑤ generate candidate k-sequences from L_{k-1} and store them in C_k
⑥ scan the database once and count the support of each candidate in C_k
⑦ look for the k-sequences in C_k with support ≥ 2 and store them in L_k
⑧ if (L_k == null) break;
⑨ Loop
⑩ for ($i=2$, $i<=k$; $k++$)
⑪ return L_i

And here is the join step:

① if there are sequence S' and S'', $S'=\{\{s_1\},\{s_3\},\{s_3\}\}$, $S''=\{\{s_2\},\{s_3,s_4\}\}$, S'' join $S'=\{\{s_1\},\{s_2\},\{s_3,s_4\}$

② if there are sequence S' and S'', $S'=\{\{s_1\},\{s_2\},\{s_3\}\}$, $S''=\{\{s_2\},\{s_3\},\{s_4\}\}$ S'' join $S'=\{\{s_1\},\{s_2\},\{s_3\},\{s_4\}\}$

for S' and S'', if $|S'-\text{first item}|=|S''-\text{last item}|$ (for example $S_2=S_2$)

then S'' can join $S' \Rightarrow S'''=S'+$ the last item of S''

The prune step:

Check [the first item of S'] + [the last item of S''] is large

if the answer is large, then store S''' into C_i

Here are the situations

① if $s_1, s_2, s_3 \cdots$ are all $|s_i| > k$ $(k=2,3\cdots)$ $(i=1,2\cdots m)$

then the number of k-sequence with support≥2 is 0

② we can count every items/sequence from L_1 to answer

if $\sum_{k=1}^{m} |s_i| = k$ (for example: $\{s_4,s_5\}$ $|s_4|+|s_5|=k$, $\{s_4,s_5\}=k$-sequence)

then $k_{max} = |s_1|+|s_3|+|s_5|+|s_1|+|s_4|+|s_5|$

$k_{min} = \min(|s_1|, |s_3|, |s_4|, |s_5|)$

for any k, $k_{min} \le k \le k_{max}$, we can use Apriori algorithm for itemsets to count the number of k-sequences easily.

③ if $k > k_{max}$, the number of k-sequence is 0

(b) NO, Apriori Algorithm can't adapting to this case

According to the lecture: Apriori Algorithm has property 1:

If an itemset S is large, then any proper subset of S must be large.

In this case, the property 1 can describe as this: If a k-sequence is large, then any proper sub-sequence of S must be large.

However, according to the definition of support of one sequence, the sub-sequences are possibly not large no matter the subquence is large or not.

For example:

$S' = \langle \{A\}, \{B\}, \{B\}, \{C\} \rangle$ Support for $\langle \{A\}, \{B\}, \{C\} \rangle = 2$

$S = \langle \{A\}, \{B\}, \{C\} \rangle$ however support for $\langle \{A\} \rangle = 1$

Q2

(a) (i) $\{C\}, \{D\}$

(ii) $\{C, D\}, \{A, C\}, \{A, D\}$

(iii) $\{A, C, D\}, \{B, E, F\}$

(b) Yes, it can be adapted in this case.

① counting: firstly, the threshold = 1.

Item	Frequency
A	2
B	1
C	3
D	3
E	1
F	1

→ sort →

Item	Frequency
C	3
D	3
A	2
B	1
E	1
F	1

⟶

(ordered) frequent items

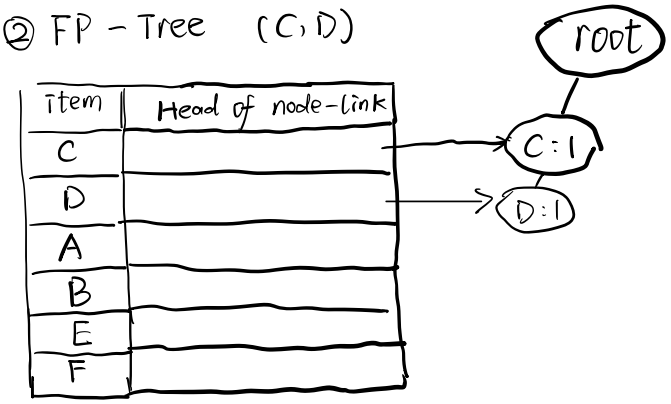
C, D

B, E, F

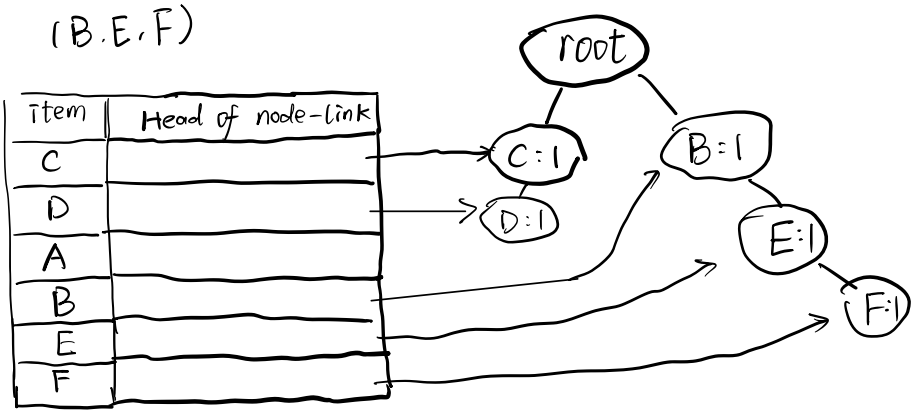
C, D, A

C, D, A

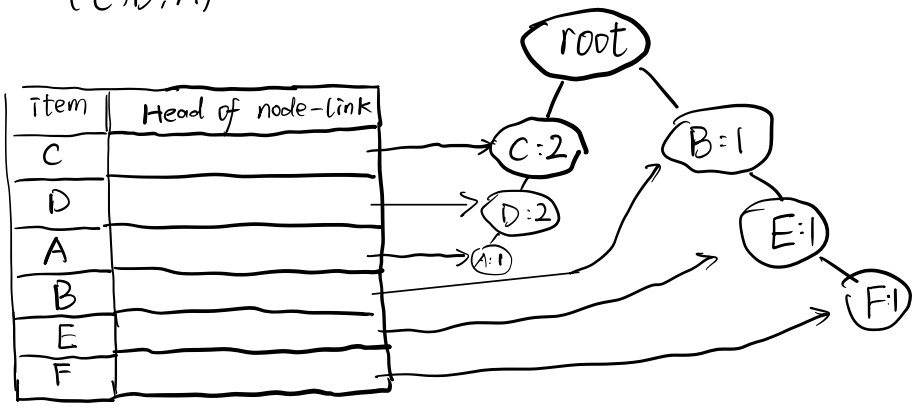
② FP - Tree (C, D)



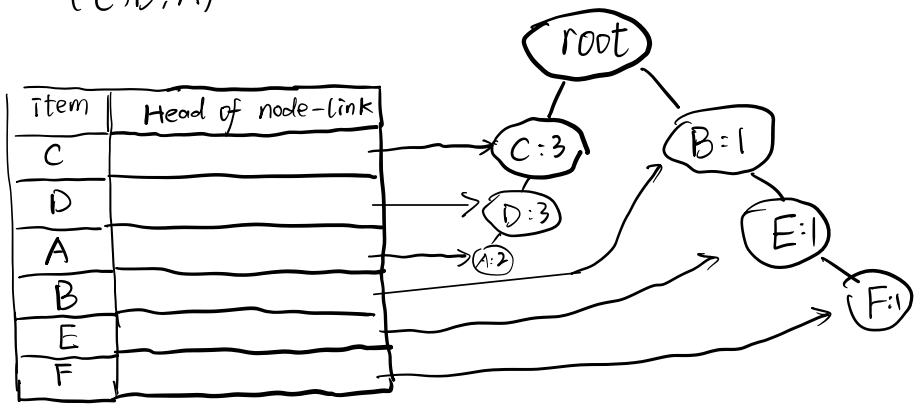
(B, E, F)



(C, D, A)



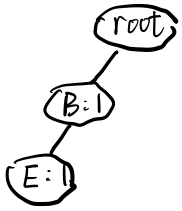
(C, D, A)



Conditional FP-tree on "F" 1

Item	Frequency
B	1
E	1

{{B:1, E:1, F:1}}



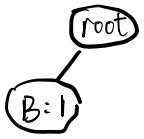
$S_{1,2} = \{F:1\}$

$S_{2,2} = \{BF:1, EF:1\}$

$S_{3,2} = \{BEF:1\}$

Condition FP-tree on "E" 1

{{B:1, E:1}}



$S_{1,2}: \{E:1, F:1\}$

$S_{2,2}: \{BF:1, EF:1, BE:1\}$

$S_{3,2}: \{BEF:1\}$

Condition FP-tree on "B" 1

{{B:1}}

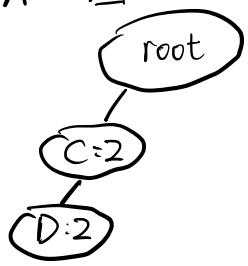
$S_{1,2}: \{B:1, E:1, F:1\}$

$S_{2,2}: \{BF:1, EF:1, BE:1\}$

$S_{3,2}: \{BEF:1\}$

Condition FP-tree on "A" 2

{{A=2; C:2, D:2}}



$\because ACD = 2$

$AC = AD = 2 > 1$, delete BF, EF, BE in $S_{2,2}$

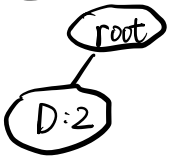
$S_{1,2}: \{A:2, B:1, E:1, F:1\}$

$S_{2,2}: \{AC:2, AD=2\}$

$S_{3,2}: \{ACD:2, BEF:1\}$

Condition FP-tree on "C" 3

{{C=3, D=3}}



$\because CD = 3$, delete B, E, F in L_1

$S_{1,2}: \{C=3, A:2\}$

$S_{2,2}$ {CD:3, AD:2, AC:2}

$S_{3,2}$ {ACD:2, BEF:1}

Condition FP-tree on "D" [3]

{{D:3}}

(root)

$S_{1,2}$: {C=3, D=3}

$S_{2,2}$: {CD:3, AD=2, AC:2}

$S_{3,2}$: {ACD:2, BEF=1}

$\therefore S_{1,2} = \{C3, \{D\}, S_{2,2} = \langle \{C,D\}, \{A,D\}, \{A,C\} ; S_{3,2} = \langle \{A,C,D\}, \{B,E,F\} \rangle$

This kind of FP-tree algorithm has "chagable" threshold. The threshold set to 1 at first.

$S_{1,2}$; $S_{2,2}$; $S_{3,2}$ are also set to 1 at first.

Also we need to three variables to record the the 2nd (if there is just one, then record the 1st) greatest value in the multi-set of the supports of all itemsets in $S_{1,2}$; $S_{2,2}$; $S_{3,2}$ respectively. I name them a_1, a_2, a_3

In this phrase, we can get the FP-tree with the support threshold of 1.

Then we need to get the FP-conditional tree with the support threshold at the stage.

① Specifically, set the variable a_i mentioned above to the i-th greatest value of the multi-set of the supports of all k-itemsets in $S_{k,i}$.

② If we can find other itemset with support greater than a_i, we add the itemset to $S_{k,i}$.

③ update a_i to be the newest i-th largest value of $S_{k,i}$

④ Remove all multi-itemset swith support smaller than a from $S_{k,i}$.

⑤ do the same things to a_2, a_3

⑥ update the threshold of support = $\min(a_1, a_2, a_3)$

(C) ① We can get top-i itemsets with greater support, while traditional FP-tree algorithm returns all frequent items with support greater than the fixed threshold

② We can have chagable support threshold, the fixed-size support-threshold is hard to set, we don't know the most appropriate value

Q3

(a) (i)

	Distance² to A(50,41)	Distance² to B(50,70)	belongs
x_1(55,50)	106	425	A
x_2(43,50)	130	449	A
x_3(55,52)	146	349	A
x_4(43,54)	218	305	A
x_5(58,53)	208	353	A
x_6(41,47)	117	610	A
x_7(50,41)	0		A
x_8(50,70)		0	B

⟹ Cluster A: x_2, x_4, x_6, x_7 (44.25, 48) ; Cluster B: x_1, x_3, x_5, x_8 mean of cluster(54.5, 56.25)

(ii)

Cluster A: x_2, x_4, x_6, x_7 (44.25, 48)

Cluster B: x_1, x_3, x_5, x_8 mean of cluster(54.5, 56.25)

(iii)

Cluster A	Cluster B	Cluster C
x_1, x_7	x_2, x_4, x_6	x_3, x_8, x_5
mean (52.5, 45.5)	(42.33, 50.33)	(54, 3, 58.3)

(iv)

Cluster A	cluster B	Cluster C	Cluster D
x_7	x_2, x_4, x_6	x_1, x_3, x_5	x_8
mean (50,41)	(42.33, 50.33)	(56, 51.67)	(50, 70)

(b) I think the answer is negative.

Here is an example:

$x_1:(0,0)$, $x_2(1,1)$, $x_3(0.5,0)$, $x_4(0.25,1)$ meanA:(0,0.5), meanB:(1,0.5)

the result of k-means: cluster A: x_1, x_4; cluster B: x_2, x_3; mean: (2.5, 1) (3.5, 5)

the result of k-means:

after x_1 is assigned, meanA:(0,0) meanB = (1, 0.5)
after x_2 is assigned, meanA:(0,0) meanB: (1,1)
after x_3 is assigned, meanA:(0.375,0) meanB: (1,1)
after x_4 is assigned, meanA: (0.375,0) meanB: (0.625, 1)
x_1, x_3 x_2, x_4

different from k-means

This is because the result obtain by sequential k-means on the presentation order of the data points. but the result of k-means are not related to the presentation order.

Q4

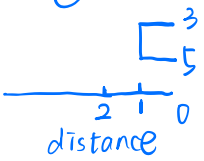
(a)

	1	2	3	4	5	6	7	8
1	0							
2	11	0						
3	5	13	0					
4	12	2	14	0				
5	7	17	1	18	0			
6	13	4	15	5	20	0		
7	9	15	12	16	15	19	0	
8	11	20	12	21	17	22	30	0

group(3,5) ⇒

	1	2	(3.5)	4	6	7	8
1	0						
2	11	0					
(3.5)	6	15	0				
4	12	2	17	0			
6	13	4	17.5	5	0		
7	9	15	13.5	16	19	0	
8	11	20	14.5	21	22	30	0

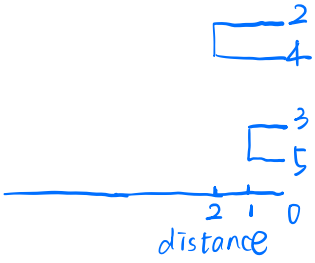
① distance



group(2,4)

	1	(2.4)	(3.5)	6	7	8
1	0					
(2.4)	11.5	0				
(3.5)	6	15.5	0			
6	13	4.5	17.5	0		
7	9	15.5	13.5	19	0	
8	11	20.5	14.5	22	30	0

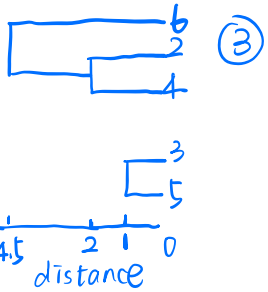
② distance



group(2,4,6)

	1	(2.4.6)	(3.5)	7	8
1	0				
(2.4.6)	12	0			
(3.5)	6	16.17	0		
7	9	16.67	13.5	0	
8	11	21	14.5	30	0

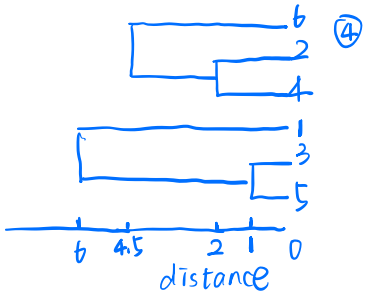
③ distance



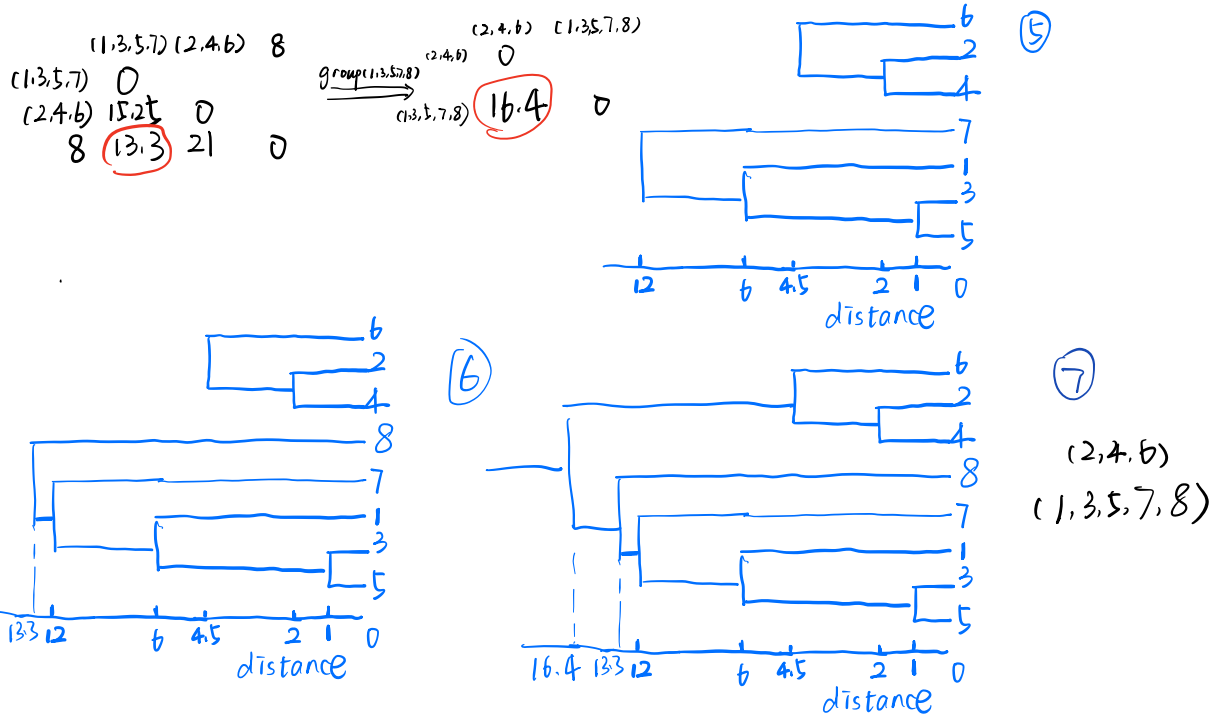
group(1,3,5)

	(1.3.5)	(2.4.6)	7	8
(1.3.5)	0			
(2.4.6)	14.78	0		
7	12	16.67	0	
8	13.3	21	30	0

④ distance



group(1,3,5,7)



(b) from the root, we can get 2 branches (2,4,6); (1,3,5,7,8)

∵ distance(8,(1,3,5,7)) = 13.3 > distance(6,(2,4)) = 4.5

∴ we get 3 branches (2,4,6) (8) (1,3,5,7)

distance(7,(1,3,5)) = 12 > distance(6,(2,4)) = 4.5

∴ finally, we get 4 branches: (2,4,6)(7)(8)(1,3,5)

(C) (i) the greatest possible number is 8, all of them

(ii) the smallest possible number is 3; (1,3,5)

(d) Yes, triangle inequality refer to $|a,b| - |b,c| \le |a,c| \le |a,b| + |b,c|$

let a,b,c now to be cluster centroids and we wish to determine the closet two centroid to join.

if we already know $|a,b|$ and $|b,c|$ and $||a,b| - |b,c|| \ge k$, then we don't need to compute the $|a,c|$, as lower bound on $|a,c|$ already exceed k, hence a,c can't be joined.

(2) the Algorithm is like this:

given that k constraint for agglomerotive methods: Two clusters whose geometric centroids are separated by a distance greater than k cannot be joined

① for $i=2$ to $n-1$

② $d_{1,j} = D(C_1, C_i)$

③ for $i=2$ to $n-1$

④ for $j=i+1$ to $n-1$ $\hat{d}_{i,j} = |d_{1,i} - d_{1,j}|$

⑤ if $\hat{d}_{i,j} > k$ then $d_{i,j} = k+1$; i,j cannot be joined
else $d_{i,j} = D(x_i, x_j)$: i,j can be joined

⑥ return $d_{i,j}$.

Q5

(a)

```
DBSCAN(DB, distFunc, ε, MinPts)
    C = 0                    # C means the cluster counter
    for each point p in database DB
        if label(p) ≠ undefined then continue      # check if processed in the inner loop
        Neighbors N = RangeQuery(DB, distFunc, P, ε)    # Find neighbors
        if |N| < Minpts then      # check density
            label(p) = Noise
            continue;
        C ++                 # next cluster
        label(p) = C         # lable initial point
        SeedSet S = N \ {p}  # Neighbors to expand
        for each point Q in S    # process every seed point Q
            if label(Q) = Noise then label(Q) = C   # update Noise to border point
            if label(Q) ≠ undefined then continue   # check whether processed before
            label(Q) = C         # label neighbor
            Neighbors N = RangeQuery(DB, distFunc, Q, ε)  # find neighbors
            if |N| ≥ MinPts then  # check if Q is a core point
                S = S ∪ N        # Add new neighbors to seed set.

RangeQuery(DB, distFunc, Q, ε)
    Neighbors N := empty list
    for each point P in database DB    # scan in database
        if distFunc(Q,P) ≤ ε then      # compute distance and check ε
            N = N ∪ {P}                # Add to result.

    return N
```

(b) $\varepsilon = 10$, $MinPt = 3$

$N(1) = \{3,5,7\} = 3 = MinPt$
$N(2) = \{4,6\} = 2 < MinPt$
$N(3) = \{1,5\} = 2 < MinPt$
$N(4) = \{2,6\} = 2 < MinPt$
$N(5) = \{1,3\} = 2 < MinPt$
$N(6) = \{2,4\} = 2 < MinPt$
$N(7) = \{1\} = 0 < MinPt$
$N(8) = \{\} = 0 < MinPt$

∴ core point : 1
border point : 3,5,7
noise point : 2,4,6,8

(c) DBSCAN visits each point of the database, possibly multiple times.
As for time complexity, we look for neighbors for each points.
$O_{overall} = n \cdot O(RangeQuery)$. As for query, we can use binary tree to store the data
so $O(RangeQuery)$ can be minimum to be $\log n$. $O($DBSCAN overall averages$) = O(n \log n)$
and the worst situation is use a list or array to store, $O($DBSCAN worst situation$) = O(n^2)$
As for database, we can easily use a list or array to store, so spatial complexity can be $O(n)$
if we use Q4 situation to store database, the spatial complexity is $O(n^2)$

(d) ① DBSCAN based on density, while hierarchical clustering is based on distance.
② DBSCAN groups together closely-packed points, not every point is part of cluster (noise).
hierarchical clustering don't.
③ DBSCAN can deal with noise and outliers. hierarchical clustering can't.
④ the border points of two clusters may be arbitrarily classified. hierarchical clustering have specific cluster.
⑤ DBSCAN has difficulty to find clusters of varying densities. but hierarchical clustering can easily find clusters of different density.