NAME: Haowei Huang
Student ID: z5247672

COMP9318 (20T1) ASSIGNMENT 1

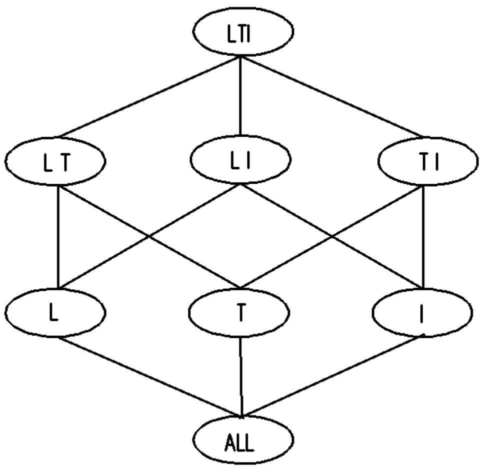
Q1: Consider the following base cuboid Sales with four tuples and the aggregate function SUM:

Location	*Time*	*Item*	*Quantity*
Sydney	2005	PS2	1400
Sydney	2006	PS2	1500
Sydney	2006	Wii	500
Melbourne	2005	XBox 360	1700

Location, Time, and Item are dimensions and Quantity is the measure. Suppose the system has built-in support for the value ALL.

(1) List the tuples in the complete data cube of R in a tabular form with 4 attributes, i.e., Location, Time, Item, SUM(Quantity)?

Using top-down algorithm to compute the complete data and get the form as below:
in the diagram, L indicates location, T indicates time and I indicates items



Location	Time	Item	Quantity
Sydney	2005	PS2	1400
Sydney	2006	PS2	1500
Sydney	2006	Wii	500
Melbourne	2005	XBox 360	1700
ALL	2005	PS2	1400
ALL	2006	PS2	1500
ALL	2006	Wii	500
ALL	2005	XBox 360	1700
Sydney	ALL	PS2	2900
Sydney	ALL	Wii	500
Melbourne	ALL	XBox 360	1700

Sydney	2005	ALL	1400
Sydney	2006	ALL	2000
Melbourne	2005	ALL	1700
ALL	ALL	PS2	2900
ALL	ALL	Wii	500
ALL	ALL	XBox 360	1700
ALL	2005	ALL	3100
ALL	2006	ALL	2000
Sydney	ALL	ALL	3400
Melbourne	ALL	ALL	1700
ALL	ALL	ALL	5100

(2) Write down an equivalent SQL statement that computes the same result (i.e., the cube). You can only use standard SQL constructs, i.e., no CUBE BY clause.

```sql
(SELECT * from q1) UNION
(SELECT 'ALL', time, item, sum(quantity) from q1 group by time, item) UNION
(SELECT location, 'ALL', item, sum(quantity) from q1 group by location, item) UNION
(SELECT location, time, 'ALL', sum(quantity) from q1 group by location, time) UNION
(SELECT 'ALL', 'ALL', item, sum(quantity) from q1 group by item) UNION
(SELECT 'ALL', time, 'ALL', sum(quantity) from q1 group by time) UNION
(SELECT location, 'ALL', 'ALL', sum(quantity) from q1 group by location) UNION
(SELECT 'ALL', 'ALL', 'ALL', sum(quantity) from q1)
;
```

(3) Consider the following ice-berg cube query:

SELECT Location, Time, Item, SUM(Quantity) FROM Sales CUBE BY Location, Time, Item HAVING COUNT(*) > 1

Draw the result of the query in a tabular form.

Location	Time	Item	Quantity
Sydney	2006	ALL	2000
Sydney	ALL	PS2	2900
ALL	ALL	PS2	2900
ALL	2005	ALL	3100
ALL	2006	ALL	2000
Sydney	ALL	ALL	3400
ALL	ALL	ALL	5100

(4) Draw the MOLAP cube (i.e., sparse multi-dimensional array) in a tabular form of (ArrayIndex, V alue). You also need to write down the function you chose to map a multi-dimensional point to a one-dimensioinal point.

firstly, we need to replace the value with corresponding index given in the functions:

Location	Time	Item	Quantity
1	1	1	1400
1	2	1	1500
1	2	3	500
2	1	2	1700
0	1	1	1400
0	2	1	1500
0	2	3	500
0	1	2	1700
1	0	1	2900
1	0	3	500
2	0	2	1700
1	1	0	1400
1	2	0	2000
2	1	0	1700
0	0	1	2900
0	0	3	500
0	0	2	1700
0	1	0	3100
0	2	0	2000
1	0	0	3400
2	0	0	1700
0	0	0	5100

function used to map a multi-dimensional point to one-dimensional:
index(L,T,I) = L + 3T + 9I and the generated one-dimensional form shown below:

index	quantity
13	1400
16	1500
34	500
23	1700
12	1400
15	1500
33	500
21	1700
10	2900
28	500
20	1700

index	quantity
4	1400
7	2000
5	1700
9	2900
27	500
18	1700
3	3100
6	2000
1	3400
2	1700
0	5100

Q2: Consider the given similarity matrix. You are asked to perform group average hierarchical clustering on this dataset.
You need to show the steps and final result of the clustering algorithm. You will show the final results by drawing a dendrogram. The dendrogram should clearly show the order in which the points are merged.

follow steps to cluster:

(1)

Column1	p1	p2	p3	p4	p5
p1	1.00	0.10	0.41	0.55	0.35
p2	0.10	1.00	0.64	0.47	0.98
p3	0.41	0.64	1.00	0.44	0.85
p4	0.55	0.47	0.44	1.00	0.76
p5	0.35	0.98	0.85	0.76	1.00

(2)

Column1	p1	p3	p4	p25
p1	1.00	0.41	0.55	0.48
p3	0.41	1.00	0.44	0.82
p4	0.55	0.44	1.00	0.74
p25	0.48	0.82	0.74	1.00

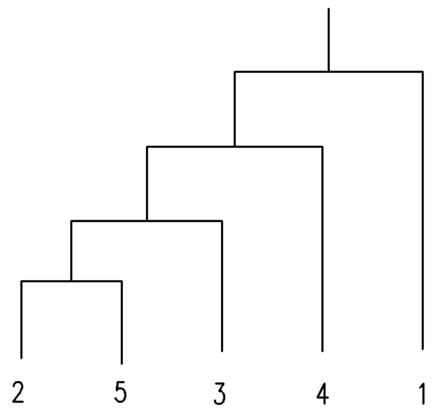
(3)

Column1	p1	p4	p235
p1	1.00	0.55	0.56
p4	0.55	1.00	0.69
p235	0.56	0.69	1.00

(4)

Column1	p1	p2345
p1	1.00	0.56
p2345	0.56	1.00

(5) Here is the dendrogram:



Q3

```
Algorithm 1: $k$-means($D$, $k$)
   Data: $D$ is a dataset of $n$ $d$-dimensional points; $k$ is the number of clusters.
1  Initialize $k$ centers $C = [c_1, c_2, \dots, c_k]$;
2  $canStop \leftarrow$ false;
3  while $canStop$ = false do
4      Initialize $k$ empty clusters $G = [g_1, g_2, \dots, g_k]$;
5      for each data point $p \in D$ do
6          $c_x \leftarrow$ NearestCenter($p$, $C$);
7          $g_{c_x}$.append($p$);
8      for each group $g \in G$ do
9          $c_i \leftarrow$ ComputeCenter($g$);
10 return $G$;
```

(1) Assume that the stopping criterion is till the algorithm converges to the final k clusters. Can you insert several lines of pseudo-code to the algorithm to implement this logic? You are not allowed to change the first 7 lines though.

Pseudo-code is shown below. Lines have been changed are followed by comment 'modified'

```
Data: D is a dataset of n d-dimensional points; k is the number of clusters.
Initialize k centers C = [c_1, c_2, ..., c_k];
canStop <- false;
while canStop = false do
    Initialize k empty clusters G = [g_1, g_2, ..., g_k];
    for each data point p ∈ D do:
        c_x <- NearestCenter(p, C);
        g_cx.append(p);
    Initialize k centers new_C = empty;      // modified
    for each group g ∈ G do:
        new_C.append(ComputeCenter(g));      // modified
    if new_C equals to C:                    // modified
        canStop = true;                      // modified
    else:                                    // modified
        C = new_C;                           // modified

return G;
```

(2) The cost of k clusters is just the total cost of each group g_i, or formally

$$cost(g_1, g_2, \ldots, g_k) = \sum_{i=1}^{k} cost(g_i)$$

cost(g_i) is the sum of squared distances of all its constituent points to the center c_i , or

$$cost(g_i) = \sum_{p \in g_i} dist^2(p, c_i)$$

dist() is the Euclidean distance. Now show that the cost of k clusters as evaluated at the end of each iteration (i.e., after Line 9 in the current algorithm) never increases. (You may assume d = 2)

```
1.  Data: D is a dataset of n d-dimensional points; k is the number of clusters.
2.  Initialize k centers C = [c_1, c_2, ..., c_k];
3.  canStop <- false;
4.  while canStop = false do
5.      Initialize k empty clusters G = [g_1, g_2, ..., g_k];
6.      for each data point p ∈ D do:            // cost0
7.          c_x <- NearestCenter(p, C);
8.          g_cx.append(p);
9.      Initialize k centers new_C = empty;      // cost1
10.     for each group g ∈ G do:
11.         new_C.append(ComputeCenter(g));
12.     if new_C equals to C:
13.         canStop = true;
14.     else:
15.         C = new_C;                           // cost2
16.
17. return G;
```

Here is the current algorithm. to Prove that the cost is never increase after each iteration, we can simply prove that: cost0 ≥ cost1 ≥ cost2. Here, cost0, cost1, cost2 are costs after each steps of a loop as shown in the pseudo-code.

After the first for loop, comparing cost0 and cost1:
The functionality of this for loop is to assign each data point to the nearest centers of the current center set. It means that each given data point find the local optimal solution.
So cost0 ≥ cost1

Then for the second for loop, comparing cost1 and cost2:
Based on the current cluster, it updates the new center set. In some way, it minimises the internal cost of each cluster, after which the algorithm enters next iteration and repeat the steps. So cost1 ≥ cost2

(3) Prove that the cost of clusters obtained by k-means algorithm always converges to a local minimal. You can make use of the previous conclusion even if you have not proved it.

From question (2), we have proved that the algorithm always converges to a lower cost. However, as (1) shows, when the newly generated center set unchanged comparing with last iteration, the algorithm would terminate. However, it doesn't mean the output is a global minimal cost.

Because the centers initialized at the beginning are randomly generated. There must be extreme examples where the algorithm finds a locally optimal solution and stops. Also, the K values is specified in advance, the global optimal solution may not exist in the current k values.