**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:**

**A screenshot of a cell phone

Description automatically generated**

**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

A close up of a logo

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| 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.**

1. (**SELECT** \* **from** q1) **UNION**
2. (**SELECT** 'ALL', **time**, item, sum(quantity) **from** q1 **group** **by** **time**, item) **UNION**
3. (**SELECT** location, 'ALL', item, sum(quantity) **from** q1 **group** **by** location, item) **UNION**
4. (**SELECT** location, **time**, 'ALL', sum(quantity) **from** q1 **group** **by** location, **time**) **UNION**
5. (**SELECT** 'ALL', 'ALL', item, sum(quantity) **from** q1 **group** **by** item) **UNION**
6. (**SELECT** 'ALL', **time**, 'ALL', sum(quantity) **from** q1 **group** **by** **time**) **UNION**
7. (**SELECT** location, 'ALL', 'ALL', sum(quantity) **from** q1 **group** **by** location) **UNION**
8. (**SELECT** 'ALL', 'ALL', 'ALL', sum(quantity) **from** q1)
9. ;

**(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 |  | index | quantity |
| --- | --- | --- | --- | --- |
| 13 | 1400 |  | 4 | 1400 |
| 16 | 1500 |  | 7 | 2000 |
| 34 | 500 |  | 5 | 1700 |
| 23 | 1700 |  | 9 | 2900 |
| 12 | 1400 |  | 27 | 500 |
| 15 | 1500 |  | 18 | 1700 |
| 33 | 500 |  | 3 | 3100 |
| 21 | 1700 |  | 6 | 2000 |
| 10 | 2900 |  | 1 | 3400 |
| 28 | 500 |  | 2 | 1700 |
| 20 | 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)

**A picture containing clock

Description automatically generated**

|  |  |  |
| --- | --- | --- |
| Column1 | p1 | p2345 |
| p1 | 1.00 | 0.56 |
| p2345 | 0.56 | 1.00 |

(5) Here is the dendrogram:

**Q3**

**A screenshot of a cell phone

Description automatically generated**

**(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’

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**:
7. c\_x <- NearestCenter(p, C);
8. g\_cx.append(p);
9. Initialize k centers new\_C = empty;   // modified
10. **for** each group g ∈ G **do**:
11. new\_C.append(ComputeCenter(g));   // modified
12. **if** new\_C equals to C:           // modified
13. canStop = **true**;   // modified
14. **else**:                    // modified
15. C = new\_C;        // modified
17. **return** G;

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

**A picture containing object, clock

Description automatically generated**

**cost(gi) is the sum of squared distances of all its constituent points to the center ci , or**

**A picture containing drawing, clock

Description automatically generated**

**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
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.