# B561 Assignment 7

Relational Programming

1. In your textbook, you have a description of the $k$-means clustering algorithm. Your task is to implement this algorithm. The input data is given in a relation `Points`(PId,x,y) where PId is an integer denoting a point and $x$ and $y$ are FLOATS given the $x$ and $y$ coordinates of that point.

   Your algorithm should work for various values of $k$.

   For more information about $k$-means clustering consult

   `https://en.wikipedia.org/wiki/K-means_clustering.`

   There, the $k$-means algorithm is given for a $d$-dimensional space. In this assignment, $d = 2$.

2. Implement the HITS authority-hubs algorithm that was discussed during class.

   The input data is given in a relation `Graph`(source INTEGER, target INTEGER) which represent the graph on which the HITS Algorithm operates. So each node in this graph will receive an authority and a hub score.

   For more information about the HITS algorithm consult

   `https://en.wikipedia.org/wiki/HITS_algorithm`
   `https://www.youtube.com/watch?v=jr3YGgfDY_E`

   An important detail of the HITS algorithm concerns the normalization of the authority vector (analogously, the hub vector). This vector needs to be normalized to have norm = 1 after each iteration step. Otherwise, the algorithm will not converge.

   Normalization of a vector of numbers can be done as follows: If $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of real numbers, then its norm $|\mathbf{x}|$ is given by the formula $\sqrt{x_1^2 + \cdots + x_n^2}$. Therefore, you can normalize the vector $(x_1, \ldots, x_n)$ by transforming it to the vector $\frac{\mathbf{x}}{|\mathbf{x}|} = (\frac{x_1}{|\mathbf{x}|} + \cdots + \frac{x_n}{|\mathbf{x}|})$. The norm of this vector will be 1.

3. Consider the following relational schemas. A tuple $(pid, spid, q)$ is in `Part_SubParts` if *spid* occurs $q$-times as a **direct** sub-part of *pid*. (For example think of a car that has 4 wheels. Furthermore, then think of a wheel that has 5 bolts.) A tuple $(pid, w)$ is in `Basic_Part` if basic part *pid* has weight $w$. A basic part is defined as a part that does not have sub-parts. In other words, the pid of the basic part does not occur in the PId column of `Part_SubParts`.

   (In the above example, a bolt would be a basic part, but car and wheel would not be basic parts.)

   The schemas of Part_SubParts and Basic_Parts are as follows:

```
Part_SubParts(PId,SId,Quantity)
Basic_Parts(PId,Weight)
```

.

You can assume that the domain of each of the attributes in these relations is INTEGER.

**Comments:**

(a) In the above example, bolt is a **direct sub-part** of wheel, but not of car. Furthermore, bolt would appear with its weight in `Basic_Parts`, but car nor wheel would appear in this relation.

In other words, only the PId's of parts that have no sub-parts in `Parts_SubParts` are in `Basic_Parts`.

(b) If the weight of a part is in `Basic_Parts`, the aggregated weight of that part is that weight. Otherwise, the aggregated weight of a part is the sum of the aggregated weights of all its **direct** sub-parts. So the weight function of a part is recursively defined.

**Example tables**: The following example is based on a desk lamp (pid = 1). Suppose a desk lamp consists of 4 bulbs (pid =2) and a frame (pid = 3), and a frame consists of a post (pid = 4) and 2 switches (pid = 5). Furthermore, we will assume that the weight of a bulb is 5, that of a post is 50, and that of a switch is 3.

Then the `Part_SubParts` and `Parts` tables would be as follows:

**Parts_SubParts**

| PID | SID | Quantity |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 3 | 1 |
| 3 | 4 | 1 |
| 3 | 5 | 2 |

**Parts**

| PID | Weight |
|---|---|
| 2 | 5 |
| 4 | 50 |
| 5 | 3 |

Then the aggregated weight of a lamp is $4 \times 5 + 1 \times (1 \times 50 + 2 \times 3) = 76$.

Write a Postgres function

```
Weight(part INTEGER) RETURNS INTEGER AS
```

..

that takes as input a part-id and returns the aggregated weight for the part with that part-id. In your solution, you can not use cursors nor the `FOR r IN SQL-query` loop statement.

4. Consider the following relational schema. A tuple $(pid, cpid)$ is in `Parent_Child` if $pid$ is a parent of child $cid$.

```
Parent_Child
-----------
|PId | SId |
------------
```

   You can assume that the domain of PId and SId is INTEGER.

   Write a Postgres program that computes the pairs $(id_1, id_2)$ such that $id_1$ and $id_2$ belong to the same generation in the `Parent-Child` relation and $id_1 \neq id_2$. ($id_1$ and $id_2$ belong to the same generation if their distance to the root in the `Parent-Child` relation is the same.)

5. Suppose you have a weighted directed graph $G = (V, E)$ stored in a ternary table named `Graph` in your database. A triple $(n, m, w)$ in Graph indicates that $G$ has an edge $(n, m)$ where $n$ is the source, $m$ is the target, and $w$ is the edge's weight. (In this problem, we will assume that each edge-weight is a positive integer.)

   Implement Dijkstra's Algorithm as a Postgres function `Dijkstra` to compute the shortest path lengths (i.e., the distance) from some input node $n$ in $G$ to all other nodes in $G$. `Dijkstra` should accept an argument $n$, the source node, and output a table `Paths` which stores the pairs $(n, d_m)$ where $d_m$ is the shortest distance from $n$ to $m$. To test your procedure, you can use the graph shown in Figure 2. The corresponding table structure for $G$ is given as the following `Graph` table.

| Source | Target | Weight |
|---|---|---|
| 0 | 1 | 2 |
| 0 | 4 | 10 |
| 1 | 2 | 3 |
| 1 | 4 | 7 |
| 2 | 3 | 4 |
| 3 | 4 | 5 |
| 4 | 2 | 6 |

   Hint: You can find the details of Dijkstra's Algorithm in the attached pdf document, but you are not required to exactly follow the pseudocode.

   When you issue CALL Dijkstra(0), you should obtain the following `Paths` table:

| Target | Distance |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 5 |
| 3 | 9 |
| 4 | 9 |

6. Write a simulation in Postgres of a MapReduce program that implements $\Pi_A(R)$ where $R(A, B)$ is a relation.

   You can assume that the domain of $A$ and $B$ is INTEGER.

7. Write a simulation in Postgres of a MapReduce program that implements the set difference of two relations $R(A)$ and $S(A)$.

   You can assume that the domain of $A$ is INTEGER.

8. Write a simulation in Postgres of a MapReduce program that implements the natural join $R \bowtie S$ of two relations $R(A, B)$ and $S(B, C)$.

   You can assume that the domain of $A$, $B$, and $C$ is INTEGER.