Max Common Connections Problem Solution

Rajesh Bhatia

Index

Solution

- **Answer:**
  - Pair which has the max common connections = **(190658, 57918)**
  - Max # of common connections = **122**

- **Language Used:** Python
- **Python Features Used:** Pandas DataFrame, Dictionary, Sets, Sorted, 3-Tuple, Generator Functions, Yield, Enumeration
- **End-to-End Execution time:** approx. 44 min

Algorithm

1. Read input CSV file into a **DataFrame** via **Generator** function which **yields** result.

2. From the DataFrame, read the rows into a **ConnectionsDict** data structure which stores the following about member connections:

    - **Key:** Key of member id

    - **Value:** Set of 1st degree connected members

3. Sort the **ConnectionsDict** data structure in the descending order of the count of elements in the value, i.e., count of elements in the set of 1st degree connections. Let's call it the **SortedConnectionsDict**. So, its (k, v) pair elements will be as follows:

    - **Key:** Key of member id

    - **Value:** Count of1st degree connections of the member

4. Declare and initialize the **Current Max 3-Tuple** as (0, 0, 0) which will store the count of the current maximum common connections between two members who are not directly connected:

    - **Current Max 3-Tuple:** (Member1, Member2, Count of common connections)

Algorithm

5. Iterate through the **SortedConnectionsDict** data structure. This will be the outer loop.

    - Get the (k1,v1) pair. Desc of (k1,v1) is provided in 3(c).

    - Break out of the for loop if the following condition is met:

        count of its 1st degree conn < the Current Max 3-Tuple count

        Reason is the SortedConnectionsDict was sorted by descending order of 1st degree connections count. So, if the count of 1st degree connections of the member < max, then the member will not have common connections > max.

    - Now, iterate through the **SortedConnectionsDict** data structure again**.** This will be the inner loop.

    - Get the (k2,v2) pair. Desc of (k2,v2) is provided in 3(c).

    - Break out of the for loop if the following condition is met:

        count of its 1st degree conn < the Current Max 3-Tuple count

        Reason is the SortedConnectionsDict was sorted by descending order of 1st degree connections count. So, if the count of 1st degree connections of the member < max, then the member will not have common connections > max.

    - Move ahead with the iteration if the following conditions are met:

        - k1 != k2

        - k1 and k2 are not directly connected

Algorithm

- To find the common connections between k1 and k2, it is enough to find the intersections of their connected members from sets v1 and v2 which can be obtained from the **ConnectionsDict** data structure.

  Hence, using set intersection, determine the common members between the member with keys k1 and k2. Clearly, k1 and k2 will be 2nd degree connected members.

  That is, we need to do:

  - Set1: set of 1st degree connected members of member with key k1
  - Set2: set of 1st degree connected members of member with key k3
  - **Common connected members = set1.intersect(set3)**

- We then compare the count of common connected members of the current iteration with the max value stored in the **Current Max 3-Tuple**:

  - if count(Common connected members) > Current Max 3-Tuple.count :

    - Store current values of k1, k3, count(set1.intersect(set3)) in the Current Max 3-Tuple:

      (k1, k3 , count(set1.intersect(set3)))

Performance Optimizations

- **First Version**
  - In the first version, the following loop was done with time complexity $O(n^2)$.
  - This will loop 200,000 * 200,000 times = 40,000,000,000 times = 40B

```
for (k1, v1 in hm.items()):
    for (k2, v2 in hm.items()):
        if ((k1 != k2) and (not directly connected(k1, k2)
            Determine set intersections of sets v1 and v2
```

Result: Out of Memory Error obtained.

Performance Optimizations

- **Second Version**
    - Then performance optimization was done to reduce the time complexity.
        - Outer loop is 200,000
        - The second loop is very less. Typically mostly in 10s or low 100s
        - The third loop is also mostly in 10s or low 100s

```
for (k1, v1 in hm.items()):      // k1: member_id, v1: level 1 connections set
    for (k2 in v1):              // k2: level 1 connections keys
        v2 = hm.get(k2, {})      // v2: level 2 connection set
        for (k3 <- v2) {         // k3: level 2 connection keys
            v3 = hm.get(k3, {})  // v3: level 2 connection set
```

Result: Out of Memory Error obtained.

Performance Optimizations

- **Final Version**

    - Finally, more optimization was done to arrive at the current algorithm which executed and gave the result quickly.

    - Most critical optimization was sorting the **ConnectionDict** in descending order of connections count and breaking out of the outer / inner loop if the number of connections of the member is less than max. At that point, there is no point in continuing with the remaining iterations / members since they will have less connections than max.

- Checks were done before adding to OutputMap:

    - k1 != k2 : both members cannot be the same
    - K1 and k2 are not directly connected
    - Only if size of set intersection was more than 0 and the new value is more than the current max was it put in **Current Max 3-Tuple.count**:

```
if ((k1 != k3) and (not directlyConnected(hm, k1, k2))):
        connSet = getConnIntersections(v1, v3)
        if len(connSet) > 0:
            if len(connSet) > Conn2HM[2]:
                Current Max 3-Tuple = (k1, k2, len(connSet))
```

Result: Worked and gave answer

Performance Optimizations

- **Final Version**

  - **Generator functions:** Input DataFrame read via **generator** function to optimize memory space.

  - Initially, the member ids and count of **set** intersections were being stored in a dictionary data structure. However, due to the data size, the process was running out of memory.

  Since the objective was just to determine the max common connections, only the maximum was then stored in a **3-Tuple**.