Parsa Yadollahi - 260869949
COMP 421 - Assignment 3

Q1:
We're going to be creating a MapReducer where the mapper takes in the person's information data set given in the question. In other words, a tuple containing the name and that individual's friend list. The reducer will take the pair of friends along with the friend list as a tuple and will return the intersection of the friend list. For this question, we're going to be using the example given in the question: Joe, (Abe, Jane, Zack) and Ali, (Sheila, Jane, Zack, Mary)

a)

**Map:**

```
    // iterate through tuples given in question (name, friendList)
    For each tuple of R:
        // Iterate through the list of friends
        For each friend of friendList:
            If name <= friend:
                return ((name, friend), (friendList))
            else:
                return ((friend, name), (friendList))
```

**Reduce:**

```
    For each tuple (friendPair, friendList)
        // If your friend list has only length 1 ⇒ means no common friends (i.e has only one
        (friends) and not (friends)(friends))
        If len(FriendList) == 1:
            return (friendPair, [])
        else:
            return (friendPair, intersection(friendList))
```

b)

c) We don't store duplicate information since the mapper sorts the key tuples. Thus, outputs where the keys are permutations are not possible since they would both relate to the same query.

d) By running over the pseudocode with the first example given in the question, the mapper will loop through all of Joe's friends and output all sorted pairs as keys with the friend list. The output is demonstrated below:

(Abe, Joe) → (Abe, Jane, Ali, Zack)x
(Jane, Joe) → (Abe, Jane, Ali, Zack)
(Ali, Joe) → (Abe, Jane, Ali, Zack)
(Joe, Zack) → (Abe, Jane, Ali, Zack)

Now using the second example given in the question and run it through the mapper once again returns the following output.

(Ali, Joe) → (Joe, Jane, Zack, Mary)
(Ali, Jane) → (Joe, Jane, Zack, Mary)

(Ali, Zack) → (Joe, Jane, Zack, Mary)
(Ali, Mary) → (Joe, Jane, Zack, Mary)

Shuffling the outputs from both these results will return the following:

(Abe, Joe) → (Abe, Jane, Ali, Zack)
(Jane, Joe) → (Abe, Jane, Ali, Zack)
(Ali, Joe) → (Abe, Jane, Ali, Zack), (Joe, Jane, Zack, Mary)
(Ali, Jane) → (Joe, Jane, Zack, Mary)
(Ali, Zack) → (Joe, Jane, Zack, Mary)
(Ali, Mary) → (Joe, Jane, Zack, Mary)
(Joe, Zack) → (Abe, Jane, Ali, Zack)

Let's take (Ali, Joe) → (Abe, Jane, Ali, Zack) (Joe, Jane, Zack, Mary) from the output and run it in the reducer.
The friend list from the output of the mapper is of length 2 meaning that there must be a common friend between the two. Since the length is greater than 1, we find the intersection of between (Abe, Jane, Ali, Zack) and (Joe, Jane, Zack, Mary) to be (Jane, Zack). The reducer will then output the pair of friends with the intersection denoted above. In other words our MapReduce will output:

(Ali, Joe) → (Jane, Zack)

Q2:
The following example is to help better understand the above MapReducers. Given the following data:

**(Hname, Province)**

(CHEO, Qc)
(Jewish General, Qc)
(Toronto hospital, On)

(**Hinsurnum, Age, Hname)**

(1, 65, CHEO)
(2, 70, Jewish General)
(3, 75, Toronto hospital)
(4, 80, CHEO)

**MapReduce1:**
Here we're going to be joining the columns hospital and patient based on the hospital name and filter patients that have age less than 60.

**Map**
The mapper will be classifying each output as either a hospital or a patient. First, we iterate through (Hname, Province) in the first loop and then iterate through (Hinsurnum, Age, Hname) in the second loop.

For each tuple of Hospital

```
        return (Hname, (Hospital, Province))
    For each tuple of Patient
        If Age > 60:
            return (Hname, (Patient, Age))
```

Examples of output we would get from the first mapper is:

```
    (CHEO, (Hospital, Qc))
    (Jewish General, (Hospital, Qc))
    (Toronto hospital, (Hospital, On))
    ...
```

And the second mapper:

```
    (CHEO, (Patient, 65))
    (CHEO, (Patient, 80))
    (Jewish General, (Patient, 70))
    …
```

**Reduce 1:**

The reduce will be joining the results of Hospital and Patient where the patient is older than 60 on the attribute HName. The pseudocode will be iterating through (Hname, vlist) where vlist is either $(Hospital,\ Province)\ or\ (Patient,\ Age)$.

```
    For each tuple from output of Mapper above:
        // create two lists containing all hospitals and patients
        hospi = []
        pati = []
        For each (type, value) in vlist:
            If (type == Patient):
                // Append the age of the patient
                pati.append(value)
            Else:
                // type = Hospital ⇒ append the Province
                hospi.append(value)
        // Iterate through all the hospitals and Patients in the list created above
        For h in hospi:
            For p in pati:
                // Merge them together
                return  (Hname, h, p)
```

The output of the pseudocode will be something like:

```
    (CHEO, Qc, 65)
    (CHEO, Qc, 80)
    (Jewish General, On, 70)
    …
```

**MapReduce2:**

Now we're going to group the MapReduce1 by province and add up the number of patients across all hospitals in their respective province. Once finished, we're going to be selecting only those provinces that have 100 patients or more.

**Map**

```
    For each (Hname, Province, Age) from the output of MapReduce1:
        return (Province, Hname)
```

Here we are simply switching the key to Province which will allow us to group by Province later

**Reduce:**

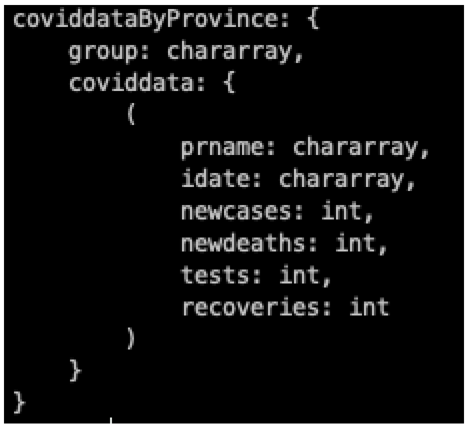
```
    numPatientsPerProvinceList = []
    For each tuple (Province, vList):
        numberOfPatients = 0
        For each v in vList:
            numberOfPatients += 1
        numPatientsPerProvince.add[(Province, numberOfPatients)]

    For each tuple in numPatientsPerProvinceList:
    If (numberOfPatients >= 100)
            return (Province, numberOfPatients)
```

This will group all provinces together and return the amount of patients per province which have more than 100 patients.

Q5:

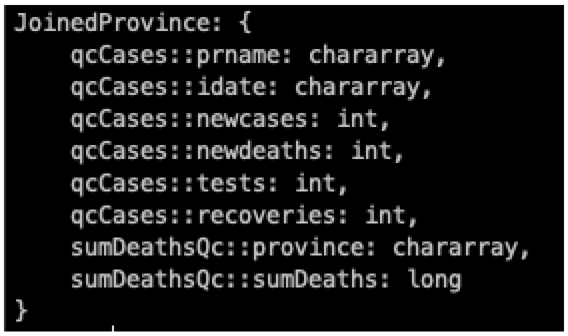
After performing the GROUP operation step, the output is the following (a screenshot of the schema from my terminal)



```
coviddataByProvince: {
    group: chararray,
    coviddata: {
        (
            prname: chararray,
            idate: chararray,
            newcases: int,
            newdeaths: int,
            tests: int,
            recoveries: int
        )
    }
}
```

Q6:

After performing the JOIN operation step, the output is the following (a screenshot of the schema from my terminal)



```
JoinedProvince: {
    qcCases::prname: chararray,
    qcCases::idate: chararray,
    qcCases::newcases: int,
    qcCases::newdeaths: int,
    qcCases::tests: int,
    qcCases::recoveries: int,
    sumDeathsQc::province: chararray,
    sumDeathsQc::sumDeaths: long
}
```