15 FEB 2023 – HWC House Water Consumption

MAPREDUCE & HADOOP

```
/**
 * Basic MapReduce Project - Mapper
 */
class Mapper1BigData extends Mapper<
                    LongWritable,
                    Text,
                    Text,
                    IntWritable> {

    protected void map(
            LongWritable key,
            Text value,
            Context context) throws IOException, InterruptedException {

        // HID,City,Country,YearBuilt
            String[] fields = value.toString().split(",");
            String city = fields[1];
            String country = fields[2];

            context.write(new Text(city + "," + country), new IntWritable(1));
    }
}


/**
 * Basic MapReduce Project - Reducer
 */
class Reducer1BigData extends Reducer<
                Text,
                IntWritable,
                Text,
                IntWritable> {

    @Override
    protected void reduce(
        Text key,
        Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {
        int sum = 0;
        String[] fields = key.toString().split(",");
        String country = fields[1];
        for(IntWritable i : values)
            sum += i.get();
        context.write(new Text(country), new IntWritable(sum));
    }
}

/**
 * Basic MapReduce Project - Mapper
 */
class Mapper2BigData extends Mapper<
        Text,
        Text,
        Text,
        IntWritable> {

    protected void map(
            Text key,
            Text value,
            Context context) throws IOException, InterruptedException {

        int v = Integer.parseInt(value.toString());
        context.write(key, new IntWritable(v));
    }
}


/**
 * Basic MapReduce Project - Reducer
 */
class Reducer2BigData extends Reducer<
        Text,
        IntWritable,
        Text,
        DoubleWritable> {

    @Override
    protected void reduce(
            Text key,
            Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {
        int sum = 0;
        int count = 0;

        for(IntWritable i : values) {
            sum += i.get();
            count++;
        }

        double mean = ((double) sum) / count;
        if(mean > 10000)
            context.write(key, new DoubleWritable(mean));
    }
}
```

SPARK

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setAppName('Exam 15 feb 2023')
sc = SparkContext(conf = conf)

housePath = "data/Houses.txt"
consumptionPath = "data/MonthlyWaterConsumption.txt"

outputPath1 = "outPart1/"
outputPath2 = "outPart2/"

# Define the rdds associated with the input files

# HID,City,Country,YearBuilt
houseRDD = sc.textFile(housePath)

# HID,Month,M3
# Month in YYYY/MM format
consumptionRDD = sc.textFile(consumptionPath)

#########################################
# PART 1
#########################################

# filter water consumptions only for years 2022 and 2021
def filter22_21(l):
    fields = l.split(",")
    month = fields[1]

    return month.startswith("2021") or month.startswith("2022")


filteredCons = consumptionRDD.filter(filter22_21)

# compute <the trimesters for each record
# to do that, we use division between integers, with months starting from 0
(january = 0, february = 1 etc)
# month // 3 = trimesterID associated to the record
# example: january // 3 = 0, february // 3 = 0, march // 3 = 0, april // 3 = 1
etc.
# We compute then an RDD with
# key = HID, trimesterID, year
# value = water consumption
```

```python
# and then sum all the values to compute the water consumed in that trimester

def HidTrimYear_Cons(l):
    fields = l.split(",")
    hid = fields[0]
    yearMonth = fields[1]
    consumption = float(fields[2])

    f = yearMonth.split("/")
    year = int(f[0])
    month = int(f[1])
    trimester = (month-1) // 3

    return ((hid, trimester, year), consumption)


waterConsPerTrim = filteredCons\
                    .map(HidTrimYear_Cons)\
                    .reduceByKey(lambda v1, v2: v1 + v2)

# perform a map, obtaining
# key = HID, trimester
# value = (m3 in 2021, m3 in 2022)
# where one of the two entries is 0
# then, use a reduceByKey to sum the entries for each year (to obtain a single
entry per house and per trimester)

def hidTrim_cons2122(t):
    hid = t[0][0]
    trim = t[0][1]
    year = t[0][2]
    m3 = t[1]

    if (year == 2021):
        return ((hid, trim), (m3, 0.))
    else:
        return ((hid, trim), (0., m3))



houseConsPerTrim = waterConsPerTrim\
                    .map(hidTrim_cons2122)\
                    .reduceByKey(lambda v1, v2: (v1[0] + v2[0], v1[1] + v2[1]))

# select the pairs with an increasing consumption (m3 2022>m3 2021)
# and map to pairs with key = HID, value = +1
# and count for each house the number of increasing consumption associated to the
same trimester
# and filter with count >= 3
housesWithIncreasingCons = houseConsPerTrim\
                            .filter(lambda t: t[1][1] > t[1][0])\
                            .map(lambda t: (t[0][0], 1))\
                            .reduceByKey(lambda v1, v2: v1 + v2)\
                            .filter(lambda t: t[1] >= 3)

# Prepare a pairRDD with
# key = hid
# value = city

def hid_city(l):
    fields = l.split(",")
    hid = fields[0]
    city = fields[1]

    return (hid, city)


houseCity = houseRDD.map(hid_city)

# join the two RDDs to compute result1
# map to HID, City
res1 = houseCity.join(housesWithIncreasingCons)\
                .map(lambda t: (t[0], t[1][0]))

res1.saveAsTextFile(outputPath1)


#########################################
# PART 2
#########################################

# compute for each year and houseID, the water consumption
# key = houseID, year
# value = water consumption

def HidYear_Cons(l):
    fields = l.split(",")
    hid = fields[0]
    yearMonth = fields[1]
    consumption = float(fields[2])
    f = yearMonth.split("/")
    year = int(f[0])

    return ((hid, year), consumption)


waterConsPerYearAndHouse = consumptionRDD\
                            .map(HidYear_Cons)\
                            .reduceByKey(lambda v1, v2: v1 + v2)

# Define windows of two consecutive year for each house
# flatMap each input to two pairs
#- key=(houseID, year  ), value=( yearly consumption, +1)
#- key=(houseID, year+1), value=(-1*yearly consumption, +1)

def elements(p):
    pairs = []

    hid = p[0][0]
    year = p[0][1]

    AnnualCons= p[1]

    # - key=(houseID, year  ), value=( yearly consumption, +1)
    pairs.append(((hid,year), (AnnualCons, 1)))

    # - key=(houseID, year+1), value=(-yearly consumption, +1)
    pairs.append(((hid,year+1), (-AnnualCons, 1)))

    return pairs


elementsWindows = waterConsPerYearAndHouse.flatMap(elements)

# Compute for each window the number of elements and the sum of the consumptions
# (each windows contains at most two values: one positive value for the current
year
# and a negative value for the previous year)
windowsElementsAndSumCons = elementsWindows\
                            .reduceByKey(lambda p1, p2: (p1[0]+p2[0],
p1[1]+p2[1]))

# Select the windows with two elements and a negative sum of consumption
(decreasing consumption)
```

```
selectedWindows = windowsElementsAndSumCons\
                    .filter(lambda p: p[1][1]==2 and p[1][0]<0)

# Map each window to (HID, +1) and remove duplicates.

def HID_1(p):
    hid = p[0][0]

    return (hid, 1)


houseAtLeastOneDecrease = selectedWindows\
                        .map(HID_1)\
                        .distinct()

# Join the previously computed RDD with houseCityRDD to get information related
to the city
# and obtain the resulting RDD with
# key = city
# value = +1
# and count the number of houses for each city for which an annual consumption
decrease was recorded
cityHouseConsDecr = houseAtLeastOneDecrease\
                    .join(houseCity)\
                    .map(lambda t: (t[1][1], 1))\
                    .reduceByKey(lambda v1, v2: v1 + v2)

# filter and keep only the cities with count
# (number of houses with at least one annual decrease) > 2
citiesToDiscard = cityHouseConsDecr\
                    .filter(lambda t:t[1] > 2)\
                    .keys()

# Select the cities with at most two houses with at least one annual decrease
res2 = houseCity\
        .map(lambda p: p[1])\
        .distinct()\
        .subtract(citiesToDiscard)

res2.saveAsTextFile(outputPath2)
```