Mahdi Sadeghi

830598035

Data Mining

Homework 1

Question 2

Question 1:
Write a Readme about hadoop installation and describe how you wrote it in the report.

Solution 1:
In writing the readme i used markdown language. The features that I used from markdown were:

- **Title, Subtitle:** you can mark a text as title with a single # and make it subtitle using two # (e.g. ## subtitle)
- **Bold:** you can make a text bold if you surround it with two * (e.g. ** bold text **)
- **Links:** you can make a text to hyperlink using [text](https://example.com)
- **Code Blocks:** you can surround a block with ``` and it will become a code block. If you specify the language after the ``` it could have language specific code highlights in github and some other interpreters.

Question 2-A:

Write a Program to Compute Column Average of input Matrix using MapReduce Method:

Solution 2-A:

Mapper: the mapper function here will simply extract matrix element info (row, column and value) from each line and emits a key, value for each of them which the key is the element's column and the value is element's value.

```
    def mapper(self, _, line):
        (i, j, v) = line.split(',')

        # yield with key=column and value=value
        yield int(j), float(v)
```

Reducer: the Reducer receives a column number as key and all the values of elements in that column as values. It should simply compute the average of these values. Because of the input file storage method (not storing elements with value 0) we cannot detect the size of column in reducer function, so I assumed the input matrix has 100 rows and the length of the columns are 100.

```
    def reducer(self, key, values):
        summation = sum(values)
        yield key, (summation / 100)
```

Question 2-B:

Write a Program to Compute Standard Deviation of each Column in the input Matrix using MapReduce Method:

Solution 2-B:

Mapper: the mapper is the same as Solution A, it just extracts element info and emits it's value with column number as key.

```
    def mapper(self, _, line):
        (i, j, v) = line.split(',')

        # yield with key=column and value=value
        yield int(j), float(v)
```

Reducer: first we should copy the input values into a list. This is because the values in the input is generator and we cannot iterate them twice. After that we compute column average like in solution A and then compute variance by subtracting each element from average and powering the result to the factor of 2. The final result of this process is the variance of elements in that particular group. We now add average^2 to variance for each zero in the column. We know how many there are from the fact that the total number of elements in a row is 100 and the number of items with non-zero value are in the values list, so the total number of zero elements are 100 - len(values). We now return sqrt(variance/100) as standard deviation with column number as key.

```
    def reducer(self, key, values):
        values_list = list(values)
        summation = sum(values_list)

        average = summation / 100

        variance = 0
        for value in values_list:
            variance += (value - average) ** 2

        for i in range(100 - len(values_list)):
            variance += average ** 2

        yield key, math.sqrt(variance / 100)
```

Question 2-C:

Write a Program to Compute Variance of each Column in the input Matrix using MapReduce Method:

Solution 2-C:

Mapper: the mapper is the same as Solution 2-B.

```
    def mapper(self, _, line):
        (i, j, v) = line.split(',')

        # yield with key=column and value=value
        yield int(j), float(v)
```

Reducer: the reducer is the same as the one in Solution 2-B but only the generated variance is yielded instead of its square root.

```
    def reducer(self, key, values):
        values_list = list(values)
        summation = sum(values_list)

        average = summation / 100

        variance = 0
        for value in values_list:
            variance += (value - average) ** 2

        for i in range(100 - len(values_list)):
            variance += average ** 2

        yield key, math.(variance / 100)
```

Question 2-D:

Write a Program to Multiply two Matrices using MapReduce Method.

Solution 2-D:

Mapper 1: First we need to know if the input line in the mapper is coming from first Matrix in equation or the second. We can detect which file is streaming into the mapper function using "map_input_file" environment variable. If this variable contains mat1.csv then we take it as the first matrix. After that if it's the first matrix, we emit it with key of it's column number and if it's the second matrix we emit a key value with key equal to elements row number. The values of these emitted key,value pairs are a tuple containing it's type (1 for first Matrix and 2 for second Matrix), it's value and it's row/column number (based on it's type). In other words For each matrix element of first Matrix (mij), produce the key value pair j, (1, i, mij). Likewise, for each matrix element njk, produce the key value pair j, (2, k, njk)[1].

```
    def mapper_1(self, _, line):
        (i, j, v) = line.split(',')

        file_name = os.environ.get('map_input_file', None)

        if 'mat1.csv' in file_name:
            yield int(j), (1, int(i), float(v))
        elif 'mat2.csv' in file_name:
            yield int(i), (2, int(j), float(v))
```

Reducer 1: first we separate first and second matrix elements in two different arrays. Now For each value that comes from First Matrix, say (1, i, mij), and each value that comes from Second, say (2, k, njk), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, mij njk.

```
    def reducer_1(self, key, values):
        values_list = list(values)

        # separate values by the matrix they came from
        tp1_values = []
        tp2_values = []

        for value in values_list:
            if value[0] == 1:
```

[1] Mining of Massive Datasets, Page 38

```
                tp1_values.append(value)
            else:
                tp2_values.append(value)

        # create key value pairs for multiplications
        for tp1_value in tp1_values:
            for tp2_value in tp2_values:
                v = tp1_value[2] * tp2_value[2]
                if v != 0:
                    yield (tp1_value[1], tp2_value[1]), v
```

Reducer 2: in the second reducer we just sum all the input values and emit them to the output with the same key they came from (the row and column of output matrix). Just before emitting we check if the value is non-zero so that we can skip emitting zero elements of output matrix.

Steps: in the steps function we define our two step algorithm using an array of two MRStep objects:

```
    def steps(self):
        return [
            MRStep(mapper=self.mapper_1, reducer=self.reducer_1),
            MRStep(reducer=self.reducer_2),
        ]
```