Thijs Quast (thiqu264)

Alexander Karlsson (aleka769)

Lab 2 - pySpark

732A54 – Big data analytics

## Question 1

Highest and lowest temperature measurements between 1950-2014:

(u'1975', (36.1, -37.0))

(u'1992', (35.4, -36.1))

(u'1994', (34.7, -40.5))

(u'2010', (34.4, -41.7))

(u'2014', (34.4, -42.5))

### 1a)

Extended program with station number included:

[x\_alkar@heffa1 ~]$ hdfs dfs -cat q1/part-00000

(u'1975', ((36.1, 86200), (-37.0, 157860)))

(u'1992', ((35.4, 63600), (-36.1, 179960)))

(u'1994', ((34.7, 117160), (-40.5, 179960)))

(u'2014', ((34.4, 96560), (-42.5, 192840)))

(u'2010', ((34.4, 75250), (-41.7, 191910)))

(u'1989', ((33.9, 63050), (-38.2, 166870)))

(u'1982', ((33.8, 94050), (-42.2, 113410)))

(u'1968', ((33.7, 137100), (-42.0, 179950)))

(u'1966', ((33.5, 151640), (-49.4, 179950)))

(u'1983', ((33.3, 98210), (-38.2, 191900)))

### 1b)

Parallelized version of the big file with 14min27 sec running time on HDFS:

(u'1975', ((36.1, u'86200'), (-37.0, u'157860')))

(u'1992', ((35.4, u'63600'), (-36.1, u'179960')))

(u'1994', ((34.7, u'117160'), (-40.5, u'179960')))

(u'2014', ((34.4, u'96560'), (-42.5, u'192840')))

(u'2010', ((34.4, u'75250'), (-41.7, u'191910')))

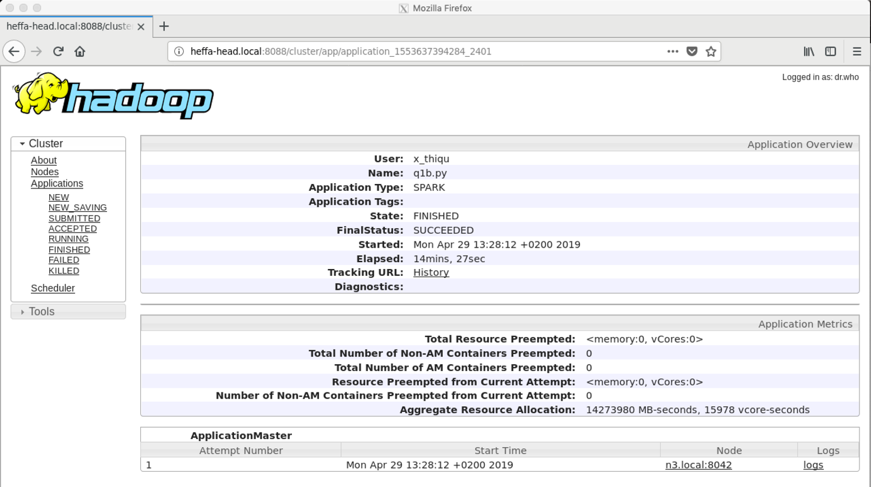
(u'1989', ((33.9, u'63050'), (-38.2, u'166870')))

(u'1982', ((33.8, u'94050'), (-42.2, u'113410')))

(u'1968', ((33.7, u'137100'), (-42.0, u'179950')))

(u'1966', ((33.5, u'151640'), (-49.4, u'179950')))

(u'1983', ((33.3, u'98210'), (-38.2, u'191900')))



The non-parallelized program outputs runtime measured in seconds. To see the results, we can check *finalResults.txt* (not shown in this document). This file is a bit unstructured, but the same results obtained when running the pySpark version is obtained (i.e the same min and max temperature measurements).

[x\_alkar@heffa2 ~]$ python q1b\_NonPar2.py

1873.21965098

A runtime of ~31 minutes is about twice the runtime as the spark program. Also, the approach is completely different. In the non-parallelized version, we simply create an empty storage object and fill it with values that corresponds to some *if*-statement. To not run out of memory, we need to read the data line by line.

One reason for the long runtime is obviously that we don’t run anything parallel. Another explanation could be the inefficiency of loops in combination with if-statements that are evaluated for every iteration (row) in our loops.

## Question 2

Number of readings for each month in the period of 1950-2014 that are higher than 10 degrees:

[x\_thiqu@heffa1 Desktop]$ hdfs dfs -cat q2a/part-00000

((2014, 7), 144824)

((2011, 7), 142587)

((2010, 7), 138997)

((2012, 7), 130691)

((2013, 7), 127454)

((2009, 7), 127430)

((2003, 7), 125604)

((2011, 8), 124676)

((2002, 7), 123771)

((2006, 8), 123482)

Repeat of the previous, this time taking distinct readings from each month:

[x\_thiqu@heffa2 Desktop]$ hdfs dfs -cat q2b/part-00000

((1973, 6), 377)

((1972, 8), 376)

((1972, 6), 375)

((1973, 5), 375)

((1972, 9), 375)

((1971, 8), 375)

((1971, 6), 374)

((1973, 9), 374)

((1971, 9), 374)

((1972, 7), 374)

## Question 3

Average monthly readings from each station in the period 1960-2014.

[x\_thiqu@heffa2 Desktop]$ hdfs dfs -cat q3\_upper/part-00000

((192840, 2014, 12), -11.825806451612905)

((192840, 2014, 11), -10.133333333333331)

((192840, 2014, 10), -1.2032258064516126)

((192840, 2014, 9), 6.196666666666666)

((192840, 2014, 8), 11.574193548387099)

((192840, 2014, 7), 16.80161290322581)

((192840, 2014, 6), 8.953333333333335)

((192840, 2014, 5), 1.8854838709677417)

((192840, 2014, 4), -2.02)

((192840, 2014, 3), -6.025806451612905)

((192840, 2014, 2), -5.4625)

((192840, 2014, 1), -19.817741935483873)

((192840, 2013, 12), -10.537096774193548)

((192840, 2013, 11), -8.615)

## Question 4

List with station number, maximum measured temperature, maximum daily precipitation filtered on temperarures between 25-30 degrees and precipitation measures in the range of 100-200 mm:

[x\_thiqu@heffa2 Desktop]$ hdfs dfs -cat q4/part-00000

[x\_thiqu@heffa2 Desktop]$

The output is empty (as it should be). Addition of code for resubmission is highlighted in green in appendix.

## Question 5

Average monthly precipitation in Östergötland between 1993-2016. Sorted ascending according to precipitation. Addition of code for resubmission is highlighted in green in appendix.

[x\_thiqu@heffa2 Desktop]$ hdfs dfs -cat q5/part-00000

((1994, 7), 0.0)

((2005, 4), 0.0)

((2009, 4), 0.0033333333333333335)

((2002, 8), 0.019354838709677424)

((2015, 10), 0.03225806451612903)

((2003, 3), 0.03870967741935484)

((2009, 4), 0.04000000000000001)

((2000, 9), 0.04285714285714286)

((2015, 10), 0.04516129032258064)

((2015, 10), 0.04838709677419355)

((2005, 3), 0.05)

((2015, 10), 0.05161290322580645)

## Question 6

Comparison of average monthly temperatures in the period 1950-2014 with long-term monthly averages (calculated between 1950-1980).

[x\_thiqu@heffa2 Desktop]$ hdfs dfs -cat q6/part-00000

((1950, 1), -2.0330490074441694)

((1950, 2), 2.2738912308955417)

((1950, 3), 2.37043883535819)

((1950, 4), 1.5510976267529655)

((1950, 5), 0.8548544197737744)

((1950, 6), -0.49158549783549965)

((1950, 7), -1.5421736276286069)

((1950, 8), 0.19116818039461236)

((1950, 9), 0.3338925081433217)

((1950, 10), -0.4721996753246751)

((1950, 11), -0.438557173678533)

((1950, 12), -0.8209155444200855)

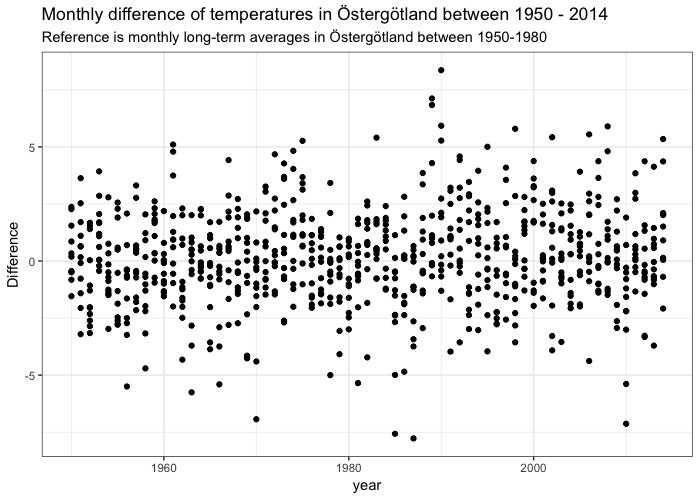
((1951, 1), 0.16808002481389472)

((1951, 2), 2.5324626594669706)

((1951, 3), -3.205367616254714)

((1951, 4), -0.087735706580367)

The results can bes een below:



Figuur 2 – Scatterplot from R

Note: we stored the results as an .xlsx-file and created the plot in R.

## Appendix

## Question 1

# Part 1

import pyspark

sc = pyspark.SparkContext(appName="Q1\_")

temp\_path = "/user/x\_thiqu/data/temperature-readings.csv"

temperature\_file = sc.textFile(temp\_path)

def max\_temperature(a,b):

if a[0]>=b[0]:

return a

else:

return b

def min\_temperature(a,b):

if a[0]<=b[0]:

return a

else:

return b

lines = temperature\_file.map(lambda line: line.split(";"))

year\_temperature = lines.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]),

float(x[3]))))

year\_temperature = year\_temperature.map(lambda x: (x[0][1], (x[1][0], x[1][1])))

year\_temperature = year\_temperature.filter(lambda x: x[0]>=1950 and x[0] <= 2014)

max\_temperatures = year\_temperature.reduceByKey(max\_temperature)

min\_temperatures = year\_temperature.reduceByKey(min\_temperature)

minmax\_temperatures = max\_temperatures.join(min\_temperatures)

minmax\_temperaturesSorted = minmax\_temperatures.sortBy(ascending=False, keyfunc = lambda k: k[1], numPartitions=1)

minmax\_temperaturesSorted = minmax\_temperaturesSorted.map(lambda x: (x[0], (x[1][0][0], x[1][1][0])))

minmax\_temperaturesSorted.saveAsTextFile("q1\_without\_stations")

# Part 2

import pyspark

sc = pyspark.SparkContext(appName="Q1\_")

temp\_path = "/user/x\_thiqu/data/temperature-readings.csv"

temperature\_file = sc.textFile(temp\_path)

def max\_temperature(a,b):

if a[0]>=b[0]:

return a

else:

return b

def min\_temperature(a,b):

if a[0]<=b[0]:

return a

else:

return b

lines = temperature\_file.map(lambda line: line.split(";"))

year\_temperature = lines.map(lambda x: (x[1][0:4], (float(x[3]), int(x[0]))))

year\_temperature = year\_temperature.filter(lambda x: int(x[0])>=1950 and int(x[0]) <= 2014)

max\_temperatures = year\_temperature.reduceByKey(max\_temperature)

min\_temperatures = year\_temperature.reduceByKey(min\_temperature)

minmax\_temperatures = max\_temperatures.join(min\_temperatures)

minmax\_temperaturesSorted = minmax\_temperatures.sortBy(ascending=False, keyfunc = lambda k: k[1])

minmax\_temperaturesSorted.saveAsTextFile("q1a")

#1b – Spark

import pyspark

sc = pyspark.SparkContext(appName="Q1\_")

temp\_path = "/user/x\_thiqu/data/temperatures-big.csv"

temperature\_file = sc.textFile(temp\_path)

def max\_temperature(a,b):

if a[0]>=b[0]:

return a

else:

return b

def min\_temperature(a,b):

if a[0]<=b[0]:

return a

else:

return b

lines = temperature\_file.map(lambda line: line.split(";"))

year\_temperature = lines.map(lambda x: (x[1][:4], (float(x[3]), x[0])))

year\_temperature = year\_temperature.filter(lambda x: int(x[0])>=1950 and int(x[0]) <= 2014)

max\_temperatures = year\_temperature.reduceByKey(max\_temperature)

min\_temperatures = year\_temperature.reduceByKey(min\_temperature)

minmax\_temperatures = max\_temperatures.join(min\_temperatures)

minmax\_temperaturesSorted = minmax\_temperatures.sortBy(ascending=False, keyfunc = lambda k: k[1], numPartitions=1)

minmax\_temperaturesSorted.saveAsTextFile("q1\_big")

#1b – Non-spark

# Module needed:

import csv

import time

# Start timer:

start = time.time()

# Filepath:

filePath = "/nfshome/hadoop\_examples/shared\_data/temperatures-big.csv"

# Import file function:

def csv\_custom\_reader(csvfile, delimiter2):

with open(csvfile, 'r') as f:

data = csv.reader(f, delimiter = delimiter2)

for row in data:

yield row

# Initialize data storage structure with dictionary:

tempDict = {}

tempDict["min"] = {}

tempDict["max"] = {}

# Create one entry for every year with nonsense values:

for yearIterable in range(1941, 2017):

tempDict["min"][yearIterable] = {"Year": int(0), "Station": int(999), "Temp": float(999)}

tempDict["max"][yearIterable] = {"Year": int(0), "Station": int(999), "Temp": float(-999)}

# Loop over file and only store values that match criterions:

for row in csv\_custom\_reader(csvfile = filePath, delimiter2 = ";"):

year = int(row[1][0:4])

stat = row[0]

temp = float(row[3])

# Only fill values that match criterion:

if tempDict["min"][year]["Temp"] > temp:

tempDict["min"][year]["Year"] = year

tempDict["min"][year]["Station"] = stat

tempDict["min"][year]["Temp"] = temp

if tempDict["max"][year]["Temp"] < temp:

tempDict["max"][year]["Year"] = year

tempDict["max"][year]["Station"] = stat

tempDict["max"][year]["Temp"] = temp

# Initialize empty dictionary for both min and max values:

finalDict = {}

# Filter out the years sought after and fill both min- and max

# temperatures in the same dictionary:

for year in tempDict["max"]:

if year >= 1950 and year <= 2014:

finalDict[year] = {"StationMin": tempDict["min"][year]["Station"],

"MinTemperarure": tempDict["min"][year]["Temp"],

"StationMax": tempDict["max"][year]["Station"],

"MaxTemperarure": tempDict["max"][year]["Temp"]}

# Write results to .txt-file:

w = open("finalResults.txt", "w")

w.write(str(finalDict))

w.close()

# End timer and print results:

end = time.time()

print(end - start)

## Code question 2

# Part 1

import pyspark

sc = pyspark.SparkContext(appName="test")

temp\_path = "/user/x\_thiqu/data/temperature-readings.csv"

temperature\_file = sc.textFile(temp\_path)

lines = temperature\_file.map(lambda line: line.split(";"))

year\_temperature = lines.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]))))

year\_temperature = year\_temperature.filter(lambda x: int(x[0][1])>=1950 and int(x[0][1])<=2014 and int(x[1]) > 10)

year\_temperature=year\_temperature.map(lambda x: (x[0][1:3], (x[1],1)))

year\_temperature = year\_temperature.map(lambda x: (x[0],1))

year\_temperature = year\_temperature.reduceByKey(lambda v1, v2: v1+v2)

year\_temperature = year\_temperature.sortBy(ascending=False, keyfunc = lambda k: k[1], numPartitions=1)

year\_temperature.saveAsTextFile("q2a")

# Part 2

import pyspark

sc = pyspark.SparkContext(appName="test")

temp\_path = "/user/x\_thiqu/data/temperature-readings.csv"

temperature\_file = sc.textFile(temp\_path)

lines = temperature\_file.map(lambda line: line.split(";"))

year\_temperature = lines.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]))))

year\_temperature = year\_temperature.filter(lambda x: int(x[0][1])>=1950 and int(x[0][1])<=2014 and int(x[1]) > 10)

year\_temperature = year\_temperature.map(lambda x: ((x[0][0:3]), (x[1])))

year\_temperature = year\_temperature.reduceByKey(lambda x, y: max(x,y))

year\_temperature = year\_temperature.map(lambda x: (x[0][1:3],1))

year\_temperature = year\_temperature.reduceByKey(lambda x, y: x +y)

year\_temperature = year\_temperature.sortBy(ascending=False, keyfunc=lambda k: k[1], numPartitions=1)

year\_temperature.saveAsTextFile("q2b")

## Code question 3

import pyspark

sc = pyspark.SparkContext(appName="test")

temp\_path = "/user/x\_thiqu/data/temperature-readings.csv"

temperature\_file = sc.textFile(temp\_path)

lines = temperature\_file.map(lambda line: line.split(";"))

year\_temperature = lines.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]),

float(x[3]))))

year\_temperature = year\_temperature.filter(lambda x: x[0][1] >= 1960 and x[0][1] <= 2014)

year\_temperature = year\_temperature.reduceByKey(lambda x, y: (min(x[0], y[0]), max(x[1], y[1])))

year\_temperature = year\_temperature.map(lambda x: ((x[0],(x[1][0]+x[1][1])/2)))

year\_temperature = year\_temperature.map(lambda x: (x[0][0:3], (x[1],1)))

year\_temperature = year\_temperature.reduceByKey(lambda x, y: ((x[0] + y[0]), x[1] + y[1]))

year\_temperature = year\_temperature.map(lambda x: (x[0], x[1][0]/x[1][1]))

year\_temperature\_max = year\_temperature.sortBy(ascending=False, keyfunc = lambda k: k[0], numPartitions=1)

year\_temperature\_min = year\_temperature.sortBy(ascending=True, keyfunc = lambda k: k[0], numPartitions=1)

year\_temperature\_min.saveAsTextFile("q3\_lower")

year\_temperature\_max.saveAsTextFile("q3\_upper")

## Code question 4

import pyspark

sc = pyspark.SparkContext(appName="test")

temp\_path = "/user/x\_thiqu/data/temperature-readings.csv"

temperature\_file = sc.textFile(temp\_path)

prec\_path = "/user/x\_thiqu/data/precipitation-readings.csv"

precipitation\_file = sc.textFile(prec\_path)

lines = temperature\_file.map(lambda line: line.split(";"))

year\_temperature = lines.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]))))

precipitation = precipitation.reduceByKey(lambda x, y: x+y)

year\_temperature = year\_temperature.map(lambda x: (x[0][0], x[1]))

def max\_temperature(a,b):

if a>=b:

return a

else:

return b

year\_temperature = year\_temperature.reduceByKey(max\_temperature)

year\_temperature = year\_temperature.filter(lambda x: x[1] > 25 and x[1] < 30)

lines2 = precipitation\_file.map(lambda line: line.split(";"))

precipitation = lines2.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]))))

precipitation = precipitation.map(lambda x: (x[0][0], x[1]))

precipitation = precipitation.reduceByKey(lambda x, y: max(x,y))

precipitation = precipitation.filter(lambda x: x[1] > 100 and x[1] < 200)

temperatures\_precipitation = year\_temperature.join(precipitation)

temperatures\_precipitation = temperatures\_precipitation.sortBy(ascending=True, keyfunc = lambda k: k[0], numPartitions=1)

temperatures\_precipitation.saveAsTextFile("q4")

## Code question 5

# Part 1

import pyspark

sc = pyspark.SparkContext(appName="test")

prec\_path = "/user/x\_thiqu/data/precipitation-readings.csv"

precipitation\_file = sc.textFile(prec\_path)

ost\_path = "/user/x\_thiqu/data/stations-Ostergotland.csv"

ostergotland\_file = sc.textFile(ost\_path)

lines = ostergotland\_file.map(lambda line: line.split(";"))

ost = lines.map(lambda x: int(x[0]))

ost = ost.collect()

lines = precipitation\_file.map(lambda line: line.split(";"))

precipitation = lines.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]))))

precipitation = precipitation.reduceByKey(lambda x, y: x+y)

precipitation = precipitation.map(lambda x: ((x[0][0:3]), (x[1],1)))

precipitation = precipitation.filter(lambda x: x[0][1] > 1993 and x[0][1] < 2016)

precipitation = precipitation.reduceByKey(lambda x, y: (x[0]+y[0], x[1]+y[1]))

precipitation = precipitation.map(lambda x: ((x[0]), (x[1][0]/x[1][1])))

precipitation = precipitation.filter(lambda x: x[0][0] in ost)

precipitation = precipitation.map(lambda x: ((x[0][1:3]),(x[1])))

precipitation = precipitation.sortBy(ascending=True, keyfunc = lambda k: k[1], numPartitions=1)

precipitation.saveAsTextFile("q5")

## Code question 6

import pyspark

sc = pyspark.SparkContext(appName="test")

ost\_path = "/user/x\_thiqu/data/stations-Ostergotland.csv"

ostergotland\_file = sc.textFile(ost\_path)

temp\_path = "/user/x\_thiqu/data/temperature-readings.csv"

temperature\_file = sc.textFile(temp\_path)

lines = temperature\_file.map(lambda line: line.split(";"))

lines\_ost = ostergotland\_file.map(lambda line: line.split(";"))

ostergotland = lines\_ost.map(lambda x: ((int(x[0]))))

ostergotland = ostergotland.collect()

year\_temperature = lines.map(lambda x: ((int(x[0]),

int(x[1].split("-")[0]),

int(x[1].split("-")[1]),

int(x[1].split("-")[2])),

(float(x[3]), float(x[3]))))

year\_temperature\_2014 = year\_temperature.filter(lambda x: x[0][1] >= 1950 and x[0][1] <= 2014)

year\_temperature\_1980 = year\_temperature.filter(lambda x: x[0][1] >= 1950 and x[0][1] <= 1980)

year\_temperature\_2014 = year\_temperature\_2014.reduceByKey(lambda x, y: (min(x[0], y[0]), max(x[1], y[1])))

year\_temperature\_1980 = year\_temperature\_1980.reduceByKey(lambda x, y: (min(x[0], y[0]), max(x[1], y[1])))

year\_temperature\_2014 = year\_temperature\_2014.map(lambda x: ((x[0],(x[1][0]+x[1][1])/2)))

year\_temperature\_1980 = year\_temperature\_1980.map(lambda x: ((x[0],(x[1][0]+x[1][1])/2)))

year\_temperature\_2014 = year\_temperature\_2014.map(lambda x: (x[0][0:3], (x[1],1)))

year\_temperature\_1980 = year\_temperature\_1980.map(lambda x: (x[0][0:3], (x[1],1)))

year\_temperature\_2014 = year\_temperature\_2014.reduceByKey(lambda x, y: ((x[0] + y[0]), x[1] + y[1]))

year\_temperature\_1980 = year\_temperature\_1980.reduceByKey(lambda x, y: ((x[0] + y[0]), x[1] + y[1]))

year\_temperature\_2014 = year\_temperature\_2014.map(lambda x: (x[0], x[1][0]/x[1][1]))

year\_temperature\_1980 = year\_temperature\_1980.map(lambda x: (x[0], x[1][0]/x[1][1]))

year\_temperature\_2014 = year\_temperature\_2014.filter(lambda x: x[0][0] in ostergotland)

year\_temperature\_1980 = year\_temperature\_1980.filter(lambda x: x[0][0] in ostergotland)

year\_temperature\_2014 = year\_temperature\_2014.map(lambda x: (x[0][1:3], (x[1],1)))

year\_temperature\_1980 = year\_temperature\_1980.map(lambda x: (x[0][1:3], (x[1],1)))

year\_temperature\_2014 = year\_temperature\_2014.reduceByKey(lambda x, y: (x[0]+y[0], x[1]+y[1]))

year\_temperature\_1980 = year\_temperature\_1980.reduceByKey(lambda x, y: (x[0]+y[0], x[1]+y[1]))

year\_temperature\_2014 = year\_temperature\_2014.map(lambda x: (x[0], x[1][0]/x[1][1]))

year\_temperature\_1980 = year\_temperature\_1980.map(lambda x: (x[0][1], x[1]))

year\_temperature\_1980 = year\_temperature\_1980.reduceByKey(lambda x, y: ((x[0]+y[0]), (x[1]+y[1])))

year\_temperature\_1980 = year\_temperature\_1980.map(lambda x: (x[0], x[1][0]/x[1][1]))

year\_temperature\_2014 = year\_temperature\_2014.map(lambda x: ((x[0][1]), (x[0][0], x[1])))

temperature\_final = year\_temperature\_2014.join(year\_temperature\_1980)

temperature\_final= temperature\_final.map(lambda x: ((x[1][0][0], x[0]), (x[1][0][1], x[1][1])))

temperature\_final = temperature\_final.map(lambda x: ((x[0]),(x[1][0]-x[1][1])))

temperature\_final = temperature\_final.sortBy(ascending=True, keyfunc = lambda k: k[0], numPartitions=1)

temperature\_final.saveAsTextFile("q6")