#######WordCount

from functools import reduce

from collections import defaultdict

import operator

def read\_file(file\_path):

with open(file\_path, 'r') as file:

sentences = file.readlines()

return[sentences.strip() for sentences in sentences]

def mapper(sentence):

words = sentence.split()

return[(word,1) for word in words]

def reducer(counts):

count\_dict = defaultdict(int)

for word, count in counts:

count\_dict[word]+=count

return dict (count\_dict)

file\_path = 'C:/Users/karti/OneDrive/Desktop/SEM 7 NOTES/BDA/practice/mapre.txt'

sentences = read\_file(file\_path)

mapped = map(mapper,sentences)

flattened = [item for sublist in mapped for item in sublist ]

reduced = reducer(flattened)

print(reduced)

########FMAlgo

data = [4, 2, 5, 9, 1, 6, 3, 7]

print("Hash functions are defined as (a\*x+b)\%c, where x is an element of the set.")

inputCount = int(input("Enter the number of hash functions: "))

abcList = []

for i in range(inputCount):

inputList = input("Enter the space-separated values of a, b and c: ").split(" ")

abcList.append([int(i) for i in inputList])

R = []

for i in abcList:

binElems = []

for j in set(data):

binElems.append(str(bin((i[0]\*j+i[1])%i[2])).split("b")[1])

r = 0

for k in binElems:

reversedCount = k[::-1]

count = 0

for i in reversedCount:

if(i=='1'):

if(count>r):

r = count

break

else:

count+=1

R.append(2\*\*r)

print("Counts recorded for each hash: ",R)

#########DGIM

import math

class DGIM:

def \_\_init\_\_(self, window\_size):

self.window\_size = window\_size

self.buckets = []

def add\_bit(self, bit, timestamp):

if bit == 1:

self.buckets.insert(0, (timestamp, 1))

self.\_merge\_buckets()

self.\_expire\_old\_buckets(timestamp)

def \_merge\_buckets(self):

i = 0

while i < len(self.buckets) - 2:

if self.buckets[i][1] == self.buckets[i+1][1] == self.buckets[i+2][1]:

new\_bucket = (self.buckets[i+1][0], self.buckets[i+1][1] \* 2)

del self.buckets[i+1:i+3]

self.buckets.insert(i+1, new\_bucket)

else:

i += 1

def \_expire\_old\_buckets(self, current\_time):

while self.buckets and self.buckets[-1][0] <= current\_time - self.window\_size:

self.buckets.pop()

def count\_ones(self, current\_time):

total = 0

for i, (timestamp, size) in enumerate(self.buckets):

if timestamp <= current\_time - self.window\_size:

break

if i == len(self.buckets) - 1:

total += size // 2

else:

total += size

return total

def display\_buckets(self):

print("Final Buckets (timestamp, size):", self.buckets)

window\_size = 24

dgim = DGIM(window\_size)

stream = [1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0]

for t, bit in enumerate(stream):

dgim.add\_bit(bit, t)

ones\_count = dgim.count\_ones(len(stream) - 1)

print(f"\nAt the last bit, number of 1's in the last {window\_size} bits: {ones\_count}")

dgim.display\_buckets()

######## R linear

> mouse.data <- data.frame(

+ weight=c(1, 2, 2, 3, 4, 4, 5, 6, 7),

+ size=c(1, 3, 1, 4, 6, 3, 3, 5, 6))

> mouse.data

> plot(mouse.data$weight, mouse.data$size)

> mouse.regression <- lm(size ~ weight, data = mouse.data)

> summary(mouse.regression)

> mouse.regression <- lm(size ~ weight, data = mouse.data)

> plot(mouse.data$weight, mouse.data$size)

> abline(mouse.regression, col = "blue")

#########R multilinear

> mouse.data<- data.frame(

+ size=c(1, 2, 1, 4, 6, 3, 3, 5, 6),

+ weight=c(1, 1, 2, 4, 4, 4, 5, 6, 6),

+ tail=c(1, 1, 1, 2, 4, 3, 3, 4, 4))

> mouse.data

> plot(mouse.data$weight, mouse.data$size)

> simple.regression <- lm(size ~ weight, data=mouse.data)

> summary(simple.regression)

> abline(simple.regression, col='red', lwd=2)

> plot(mouse.data)

> multiple.regression <- lm(size ~ weight +tail, data=mouse.data)

> summary(multiple.regression)

##########R log

//Logistic regression

study\_hours <- c(1, 2, 3, 4, 5, 6)

pass\_exam <- c(0, 0, 0, 1, 1, 1) # 0 = Fail, 1 = Pass

data <- data.frame(study\_hours, pass\_exam)

logistic\_model <- glm(pass\_exam ~ study\_hours, data = data, family = binomial)

summary(logistic\_model)

new\_data <- data.frame(study\_hours = seq(1, 6, 0.1))

new\_data$predicted\_prob <- predict(logistic\_model, newdata = new\_data, type = "response")

ggplot(data, aes(x = study\_hours, y = pass\_exam)) +

geom\_point(size = 3) + # Add points

geom\_line(data = new\_data, aes(y = predicted\_prob), color = "blue") + # Add regression line

labs(title = "Logistic Regression: Probability of Passing vs Study Hours", x = "Study Hours", y = "Probability of Passing") +

ylim(-0.1, 1.1) # Set y-axis limits

// mongo db cmmd

show dbs Lists all the databases available on the MongoDB server.

use <database\_name> Switches to the specified database; creates it if it doesn't exist.

show collections Lists all the collections in the current database.

db.<collection>.insertOne({...}) Inserts a single document into the specified collection.

db.<collection>.find() Retrieves all documents from the specified collection.

db.<collection>.find({key: value}) Retrieves documents that match the specified query.

db.<collection>.updateOne({...}) Updates a single document in the collection based on a query.

db.<collection>.deleteOne({...}) Deletes a single document from the collection based on a query.

db.<collection>.countDocuments() Returns the number of documents in the specified collection.

db.<collection>.createIndex({...}) Creates an index on the specified field(s) to improve query performance.

db.<collection>.drop() Deletes the specified collection and all its documents.

db.dropDatabase() Deletes the current database.

exit Exits the MongoDB shell (mongosh).

// HADOOP COMMANDS

hdfs dfs -help Displays a list of available HDFS commands and their usage.

hdfs dfs -ls <path> Lists files and directories in the specified path.

hdfs dfs -mkdir <path> Creates a new directory at the specified path.

hdfs dfs -ls -r <path> Recursively lists files and directories under the specified path.

hdfs dfs -get <src> <dest> Copies files or directories from HDFS to the local filesystem.

hdfs dfs -put <src> <dest> Uploads files or directories from the local filesystem to HDFS.

hdfs dfs -cat <path> Displays the contents of a file at the specified path.

hdfs dfs -tail <path> Displays the last few bytes of a file at the specified path.

hdfs dfs -rm <path> Removes (deletes) files or directories at the specified path.

// Using -put and -get commands to move a file to and from local system and hdfs, then deleting the file from hdfs using -rm

hdfs dfs -put src(D:\a.txt /mydir)

hdfs dfs -ls /mydir

hdfs dfs -cat /mydir/a.txt

hdfs dfs -get /mydir/a.txt D:\temp\

dir D:\temp\

Volume in drive D is Data

Volume Serial Number is 40A8-1731

Directory of D:\temp

hdfs dfs -rm /mydir/a.txt

Deleted /mydir/a.txt

hdfs dfs -ls /mydir

######R Basic Queries

install.packages("dplyr")

library("dplyr")

data(mtcars)

query\_condition<-mtcars%>%

filter(mpg>20)

print(query\_condition)

install.packages("dplyr")

library("dplyr")

data(mtcars)

ad\_hoc\_query\_result<-mtcars%>%

filter(hp>150) %>%

arrange(desc(hp))

print(ad\_hoc\_query\_result)

install.packages("dplyr")

library("dplyr")

data(mtcars)

perform\_continuous\_query<-function(data, cylinder\_count=6, interval=5){

for(i in 1:5){

result<-data%>%

filter(cyl==cylinder\_count) %>%

arrange(desc(wt))

print(paste("Period:",i))

print(result)

Sys.sleep(interval)

}

}

perform\_continuous\_query(mtcars, cylinder\_count=6)